Noname manuscript No.
(will be inserted by the editor)

On Efficient 3D Object Retrieval (Technical Report)

Hao Liu · **Raymond Chi-Wing Wong**

Received: date / Accepted: date

Abstract Due to the growth of the 3D technology, digital 3D models represented in the form of point clouds have attracted a lot of attention from both industry and academia. In this paper, due to a variety of applications, we study a fundamental problem called the *3D object retrieval*, which is to find a set of 3D point clouds stored in a database that are similar to a given query 3D point cloud. To the best of our knowledge, solving the problem of 3D object retrieval *efficiently* remains unexplored in the research community. In this paper, we propose a framework called C_2O to find the answer efficiently with the help of an index built on the database. In most of our experiments, C_2O performs up to 2 orders of magnitude faster than all adapted algorithms in the literature. In particular, when the database size scales up to 100 million points, C_2O answers the 3D object retrieval within 10 seconds but all adapted exact algorithms need more than 1000 seconds.

Keywords Point Cloud · 3D Object Retrieval

1 Introduction

Recently, digital 3D models represented in the form of point clouds have attracted a lot of attention from both industry and academic due to the increasing popularity of low-cost 3D depth sensors and LiDAR sensors [60, 34]. In industry, both hardware and software are now using 3D point clouds. For hardware, some recent mobile devices like iPhone 15 Pro, iPad Pro and Samsung's Galaxy S22 use LiDAR sensors or other technology to obtain 3D point clouds easily. For software, Google is now using 3D point clouds in their products. One example is that some of the recent 3D street view images in Google Maps were generated from 3D point clouds taken by LiDAR sensors starting from 2017 [2]. Another example is that Google mobile apps started to include a library about 3D point clouds (called "PointCloud") in their software development kit "ARCore" for the augmented reality (AR) application [3]. Due to the recent LiDAR technology usage in both software and hardware, there are different industrial domains using 3D point clouds. One example is the civil engineering domain which is using 3D point clouds in their building information models (BIM) for design, construction and maintenance [35]. Another example is the property agency domain which is currently using 3D point clouds for visualizing a 3D first-person view of a property in the virtual reality (VR)-like application [4]. In academia, there are a lot of research projects on point clouds. Some examples are 3D reconstruction [21], indoor robotics [51], autonomous driving [30] and VR/AR [56]. Thus, studying problems on point clouds could be interesting in the database community (especially, the spatial database community). In Figure 1, there are five 3D objects each of which is represented by a *point cloud* where a point cloud is defined to be a set of points in a 3D space.

In most (if not all) of the applications on 3D point clouds, *3D object retrieval* is one fundamental problem. Specifically, we are given a set $\mathscr{P}$ of a number of point clouds (each representing a 3D object). Given a query point cloud Q and a user parameter δ where δ is a non-negative real number, we want to find a set of point clouds in $\mathscr{P}$ such that for each point cloud P in the set, Q is *similar* to a portion of P. More formally, the *distance* between Q and a portion of P is at most δ. It is worth mentioning that under our problem setting, Q could be similar to the entire

H. Liu
E-mail: hliubs@cse.ust.hk

R. C.-W. Wong
E-mail: raywong@cse.ust.hk

H. Liu · R. C.-W. Wong
Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong, China

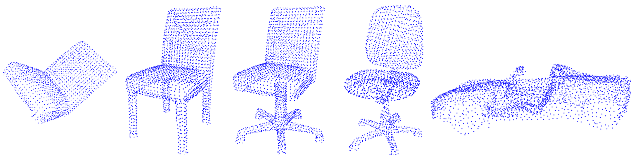


Fig. 1 A Motivating Example

P (which is a special case of a (full) portion of P). If δ is set to 0, it is an exact match between Q and a portion of a point cloud P. If δ is set to a value larger than 0, it is a similarity search problem between Q and a portion of a point cloud P. Consider Figure 1 where $\mathscr{P}$ contains Objects 1–4. Suppose that our query point cloud is shown in the figure and δ is set to a positive value near to 0. It is easy to verify that our 3D object retrieval returns Object 1 and Object 2 in the answer (not Object 3 and Object 4) since both Object 1 and Object 2 contain the back rest part which is similar to the query but both Object 3 and Object 4 do not.

There are a lot of applications about 3D object retrieval.

- (1) *3D Object Database Search:* In BIM, due to the growth of 3D printing in the building construction industry, various construction companies producing building components provide a database $\mathscr{P}$ containing all 3D models each representing a building component for other companies to search for a particular 3D model Q [47], which becomes more common nowadays [47]. In machine construction, similarly, the 3D printing technology boosts the growth of the 3D object database storing all machine components [1].
- (2) *Unmanned Vehicle Localization:* When a 3D indoor scene is constructed in the form of 3D point cloud (which can be regarded as a database point cloud in $\mathscr{P}$), an unmanned vehicle could determine its location by using its LiDAR sensors to obtain a point cloud as a query point cloud Q which is used to find which portion of the whole 3D indoor scene point cloud [17]. This becomes much more important in either indoor or a territory outside the Earth (e.g., Moon and Mars) when GPS signals are inaccurate or unavailable [19].
- (3) *Scientific Database Search:* In a scientific database $\mathscr{P}$ like a celestial database containing all stars in the galaxy, scientists would like to find a particular star pattern Q in the database or a list of regions in the galaxy which have star patterns similar to a given star pattern [53].
- (4) *Medical Implant Search:* In recent medical applications, given a database $\mathscr{P}$ of 3D point clouds each representing an implant, which can be used for organ transplantation, it is essential to search for an optimal implant that best fits the patient's need of organ transplantation, by obtaining a query point cloud Q representing the previous organ from the patient and find one in $\mathscr{P}$ that is similar to Q [32].

To the best of our knowledge, there are two branches of existing studies in the literature which are closely related to our problem. The first branch is called *registration* [7,18,50,37,25,26] which is different from our problem. There are two types of registration, namely *full registration* [7,18,50,37] and *partial registration* [25,26]. In full registration, we are given two point clouds, namely P_1 and P_2, where the total number of points in P_1 is roughly equal to the total number of points in P_2. The problem of full registration is to determine whether P_1 could be translated or rotated such that the *whole* resulting point cloud is roughly equal to P_2. If yes, it will return the resulting point cloud (from P_1). Partial registration is similar to full registration but partial registration is to determine whether P_1 could be translated or rotated such that a *portion* of the resulting point cloud is roughly equal to a *portion* of P_2. However, existing studies about partial registration require that this portion should be large (e.g., more than 60%). It is worth mentioning that the registration problem (either full registration or partial registration) is different from our 3D object retrieval problem. Firstly, it only considers "mapping" *two* given 3D point clouds but we consider how to "search" for a given query point cloud in a database involving *many* 3D point clouds. Secondly, partial registration considers "mapping" two given 3D point clouds with *large* portions but we consider "mapping" two 3D point clouds (one from the query 3D point cloud and the other from the database) in any arbitrary portion (which could be large or small).

The second branch is called *similarity search* (or called *shape matching*) [55,59,14,36], which is to find a set of *different* portions of a given database point cloud representing a 3D scene such that each portion "looks" similar to a query object Q (e.g., the back rest part of a chair) represented in the form of a point cloud. Specifically, each portion of a given database point cloud is encoded as an *embedding vector* by a machine learning model M and the query object Q is also encoded as an embedding vector by the same model M. These studies about similarity search are to return a set of portions whose embedding vectors have their Euclidean distance to the query embedding vector at most a given threshold value. However, since this existing problem adopts the concept of "embedding vectors" derived from 3D point clouds which could capture the shape of 3D objects *roughly* not *exactly*, the 3D point clouds returned in the output may not really be a real chair, and thus, the accuracy is not high. For instance, when the query object is the back rest part of a chair (as shown in Query of Figure 1), a car (as shown in Object 4 of Figure 1) is unfortunately returned by [55] as an output in our experiment (because the shape of the car is *wrongly* captured into an embedding vector similar to the embedding vector of the query object), but the expected outputs are chairs with similar back rest parts (e.g., Object 1–3 of Figure 1). It is worth mentioning that there are some recent studies [55,59] in this branch with problem named "object retrieval", which actually refers to the same problem as similarity search. Hence, their "object retrieval" problem is still different from our "object retrieval" problem definition and has the same inaccuracy issue. Moreover, [55,59] do not consider rotation/translation and thus give incorrect results easily. For example, [55,59] fail to return the expected chairs (e.g., Object 1–3 of Figure 1) for a query with the *rotated* back rest part of a chair (as shown in Query of Figure 1).

Unfortunately, none of the adapted and existing algorithms could solve our 3D object retrieval problem well. Firstly, although no existing algorithms, originally designed for registration, directly solve the 3D object retrieval problem, we adapt these existing algorithms, namely Super4PCS [39] and GoICP [58], for our problem. All of these adapted algorithms suffer from one of the following problems. The first problem is low efficiency, particularly when the database size is large. The second problem is a very bulky index size (for those adapted algorithms using indices). Secondly, existing algorithms originally designed for similarity search, namely PointNetVLAD [55] and PCAN [59], are not accurate as described previously.

Motivated by this, we propose a novel and concise representation called *donut* to denote some important "features" of a 3D point cloud so that whenever we want to compare two point clouds, instead of comparing the coordinates of all the points in one point cloud with the coordinates of all the points in another point cloud, we just need to compare the donut representation of one point cloud with the donut representation of another point cloud. Since the donut representation of a point cloud is concise, it is easier to do a comparison. Specifically, the donut representation has the following advantages: *translation-invariant* and *rotation-invariant*. When a point cloud is compared with another point cloud, since these 2 point clouds are in 2 different spaces, we have to perform a translation operation and a rotation operation (which are very time-consuming). In Figure 1, the point cloud indicated by "Query" has to perform these operations so that it can be compared with each of the 4 database point clouds in the same space. The donut representation of a point cloud is said to be *translation-invariant* (*rotation-invariant*) if no matter whether we perform a translation (rotation) operation on the point cloud, the donut representation does not change. It is important that the donut representation is translation-invariant and rotation-invariant because it could avoid the time-consuming operations of translation and rotation.

When both the query point cloud Q and all database point clouds are encoded in the form of the donut representation, it is more efficient to find a list of database point clouds which are similar to the query point cloud.

In this paper, we propose an algorithm called C_2O which involves 2 phases, namely the preprocessing phase and the query phase. In the preprocessing phase, C_2O first builds an index on all database point clouds based on the donut representation. In the query phase, given a query point cloud Q, C_2O generates a *small* set of candidates denoting the possible "features" from Q by our novel pruning strategies, and finds a set of point clouds in the database efficiently based on this small set with the help of the index.

Our contributions are as follows.

- Firstly, we study the *efficient* 3D object retrieval problem, which, to the best of our knowledge, remains unexplored since efficiency is the main focus in the database community and has not been studied extensively in the literature.
- Secondly, we propose a novel and concise representation called donut to effectively capture the important features of 3D point clouds so that retrieving similar 3D objects is very effective and efficient due to its translation-invariant property and its rotation-invariant property.
- Thirdly, based on the donut representation, we propose an index-based algorithm called C_2O to find a set of similar 3D objects efficiently.
- Fourthly, we conducted comprehensive experiments to show that our proposed algorithm C_2O is very effective and efficient. In particular, our proposed algorithm is 1-3 orders of magnitude faster than the adapted algorithms. We scale the database size up to 100 million points, a large data volume that has been applied in various real-world applications (e.g. localization [26], object recognition [31], and 3D reconstruction [43]), and we outperform all adapted algorithms since our proposed algorithm took less than 10 seconds for retrieving 3D objects but all adapted exact algorithms took more than 1000 seconds, which is not acceptable in real-life applications. Besides, the f-measure of our proposed algorithm is 100% but the f-measure of the existing algorithms originally designed for similarity search is around or less than 15% (with both precision and recall less than 25%), which is not acceptable too.

In the rest of this paper, we give the formal definition of our 3D object retrieval in Section 2. In Section 3, we first describe our framework C_2O. Then, we present the detail of our algorithm C_2O in Section 4. In Section 5, we introduce the relevancy of existing problems and existing algorithms. In Section 6, we present our experimental results. Finally, we conclude our paper in Section 7.

2 Problem Definition

Consider a 3-dimensional space. A *point cloud* is defined to be a set of 3-dimensional points where each point is rep-

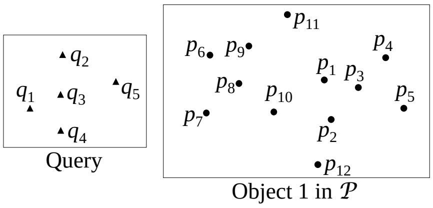


Fig. 2 Examples of Database and Query

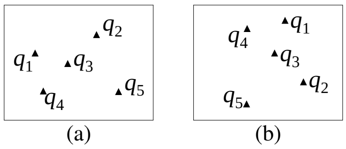


Fig. 3 Examples of Rotated Query Point Clouds: (a) Rotated with 45° Clockwise, (b) Rotated with 135° Clockwise

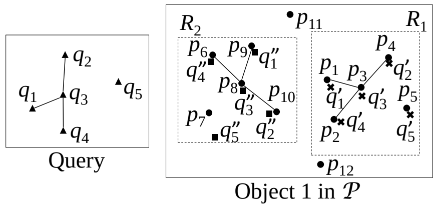


Fig. 4 Examples of Transformation and Retrieval

resented in a 3-dimensional xyz coordinate. The size of a point cloud is defined to be the total number of points in this point cloud. For example, in Figure 1, all the points in "Query" form a point cloud and all the points in Object 1 form another point cloud. We are given a database $\mathscr{P}$ which contains $|\mathscr{P}|$ point clouds. Each point cloud in $\mathscr{P}$ is associated with an ID. In the application of 3D object database search, each point cloud in the database denotes a 3D object. In the application of unmanned vehicle localization, each point cloud in the database denotes a 3D scene. We define the size of a database, denoted by n, to be the sum of the sizes of all point clouds in the database. Let us create a toy example as shown in Figure 2 to show our concept more easily where we have one query point cloud and we have the database $\mathscr{P}$ containing many objects but we just show Object 1 in $\mathscr{P}$ in the figure. Here, we illustrate the example in a two-dimensional space (for the ease of illustration) but the same illustration can be applied to a three-dimensional space.

To compare two point clouds in two different coordinate systems, we have to perform a *rigid transformation* on one point cloud so that the transformed point cloud is in a consistent coordinate system with the other point cloud for comparison. A rigid transformation Θ on a point cloud is formed by a rotation operation and a translation operation such that the geometric structure of the point cloud is preserved. In geometry, a rotation operation is denoted by a 3D rotation 3×3 matrix R and a translation is denoted by a translation 3-dimensional vector t. A rigid transformation Θ is represented by a 2-tuple (R,t). Given a point p (represented in a 3-dimensional vector) and a transformation $\Theta = (R,t)$, the *transformed point* of p wrt Θ, denoted by $T_\Theta(p)$, is defined as follows: $T_\Theta(p) = Rp + t$. That is, point p is rotated with the rotation matrix R and is translated with the translation vector t. Given a point cloud P, we define the *transformed point cloud* of P wrt Θ, denoted by $T_\Theta(P)$, to be $\{T_\Theta(p) \mid p \in P\}$. Figure 3(a) shows the query point cloud from Figure 2 rotated with 45^o clockwise and Figure 3(b) shows the query point cloud from Figure 2 rotated with 135^o clockwise.

In our problem, we need to compare a query point cloud with a database point cloud. We just need to do a rigid transformation on one point cloud, which is chosen as the query point cloud in this paper. Figure 4 shows the rotated query point cloud in Figure 3(a) in the coordinate system of Object 1 which undergoes a translation operation (indicated in a dashed rectangle R_1). Similarly, Figure 4 also shows the rotated query point cloud in Figure 3(b) in the coordinate system of Object 1 which undergoes a translation operation (indicated in a dashed rectangle R_2).

Let us consider how we define the distance between two point clouds in the same coordinate system for easier illustration. After that, we will consider the case when the two point clouds are in different coordinate systems. Consider two point clouds in the same coordinate system, namely Q and P. For each point q in Q, we define the *correspondence point* of q on P, denoted by $corr(q,P)$, to be the point in P that has the smallest Euclidean distance to q (which is a common definition in the literature [11]). We denote the Euclidean distance between point q and point p by $\|q,p\|$. The distance between Q and P in the same coordinate system, denoted by $dist_{same}(Q,P)$, is defined as follows based on the concept of the *average* L_2-norm.

$$dist_{same}(Q,P) = [\tfrac{1}{|Q|} \textstyle\sum_{q \in Q} \|q, corr(q,P)\|^2]^{\frac{1}{2}} \tag{1}$$

Note that the above distance definition to compare two point clouds Q and P has been widely applied in [58, 11,7,39]. While other distance definitions (e.g., the *exact distance* [48,13] and the *Hausdorff distance* [20]) exist in the literature, they have obvious issues. For instance, the exact distance (which returns 0 if Q and P are exactly equal, and 1 otherwise) can only tell whether Q and P are the same, but cannot measure how similar they are, and the Hausdorff distance (which returns the maximum $corr(q,P)$ for any $q \in Q$) is easily affected by an outlier in Q far away from its correspondence point on P. Our applied distance definition avoids the above issues by using the average L_2-norm, which could effectively measure the similarity of two point clouds even when noise and outliers are present [7].

Consider back Figure 4. We know that the rotated query point cloud Q' in R_1 is in the same coordinate system of Object 1. Suppose that P is the point cloud denoting Object 1. In this figure, we know that the correspondence point of q_1' on P is p_1 (i.e., $corr(q_1',P) = p_1$) and the correspondence point of q_2' on P is p_4 (i.e., $corr(q_2',P) = p_4$). That is, $corr(q_1',P) = p_1$ and $corr(q_2',P) = p_4$. After we find the correspondence point of each point in the rotated query point cloud Q', we can compute the distance between Q' and P (i.e., $dist_{same}(Q',P)$). Since *visually*, each query point in Q'

is very close to a point in P, the distance between Q' and P is very small (e.g., near to 0). It is easy to have a similar conclusion for the rotated query point cloud Q'' in R_2. Note that since q''_5 is a little bit far away from its closest point in P (i.e., p_7)(compared with other points in Q''), we derive that in general, $dist_{same}(Q'',P) > dist_{same}(Q',P)$.

However, as mentioned before, typically, two given point clouds are usually not in the same coordinate system. We have to perform a rigid transformation on one point cloud (in our case, the query point cloud). Consider two point clouds in different coordinate systems, namely Q and P. Given a transformation Θ on Q, the distance between Q and P in different coordinate systems wrt Θ, denoted by $dist_{diff}(Q,P|\Theta)$, is defined as follows.

$$dist_{diff}(Q,P|\Theta) = [\frac{1}{|Q|}\sum_{q' \in T_\Theta(Q)} \|q', corr(q',P)\|^2]^{\frac{1}{2}} \quad (2)$$

It is the same as Equation 1 except that we include the rigid transformation Θ on Q only in the above equation.

Let Θ_o be the optimal rigid transformation in a set $\mathscr{T}$ of all possible transformations of Q such that Equation 2 is minimized. That is, $\Theta_o = \arg\min_{\Theta \in \mathscr{T}} [\frac{1}{|Q|}\sum_{q' \in T_\Theta(Q)} \|q', corr(q',P)\|^2]^{\frac{1}{2}}$. We overload symbol $dist_{diff}(\cdot,\cdot)$ and define the distance between Q and P in different coordinate systems, denoted by $dist_{diff}(Q,P)$, to be the following.

$$dist_{diff}(Q,P) = [\frac{1}{|Q|}\sum_{q' \in T_{\Theta_o}(Q)} \|q', corr(q',P)\|^2]^{\frac{1}{2}} \quad (3)$$

In the following, for the sake of simplicity, when we write $dist(Q,P)$, we mean $dist_{diff}(Q,P)$.

Definition 1 (3D Object Retrieval) Given a query point cloud Q and a user parameter δ where δ is a non-negative real number, we want to find a set of point clouds in $\mathscr{P}$ such that for each point cloud P in the set, $dist(Q,P) \leq \delta$.

In Table 1, we summarize the commonly used notations in this paper.

3 Framework C_2O

In our 3D object retrieval problem, we have to check whether a given query point cloud Q and a given point cloud P in the database $\mathscr{P}$ have $dist(Q,P) \leq \delta$. We propose our framework called C_2O which involves the following 3 major steps to complete this goal.

- **Step 1 (Coarse Transformation):** We first perform a transformation Θ on Q based on *some* "representative" points of Q so that the transformed point cloud of Q and the point cloud P are in the same coordinate system. Since the transformation Θ is based on some "representative" points of Q (not *all* points of Q), we call Θ a *rough* or *coarse* transformation. The reason to have a coarse transformation is that finding an *optimal* transformation is costly, because it involves a time-consuming search that considers all possible transformations involving all points [58]. In a typical experimental setting (e.g., with the database size 1M and the query size 100), finding an optimal transformation from *scratch* takes over 1000 seconds but finding a rough transformation from some points in Q takes about 0.5 second only.
- **Step 2 (Complete Transformation):** We then perform a *complete* transformation Θ_o on Q based on *all* points of Q according to the initial coarse transformation Θ so that the transformation on Q is optimal (in terms of Equation 2). Note that with the help of the initial transformation Θ, finding the optimal transformation is much faster. In our experiment (e.g., a typical setting described above), it takes 0.9 second only.
- **Step 3 (Object Retrieval):** After we obtain the optimal transformation Θ_o on Q, we can compute $dist_{diff}(Q,P|\Theta_o) (= dist(Q,P))$ easily and check whether $dist(Q,P) \leq \delta$. If yes, P will be in the answer of our 3D object retrieval.

The most challenging step in framework C_2O is coarse transformation, because finding a "close-to-optimal" transformation based on some "representative" points of Q could be costly (though much cheaper than finding the optimal one) since searching "representative" points blindly may not help to improve the efficiency of this step.

One may ask the following question. How could we obtain some "representative" points of Q in Step 1 so that we could execute coarse transformation efficiently? In this paper, we propose a novel and concise representation of a given point cloud called the *donut* representation. Specifically, each point in a given point cloud can be encoded in a representation called the *relative-distance representation* which is represented in the form of a 6-dimensional tuple where each dimension in this tuple is a real number For each point $p_{(1)}$ in P, we find 3 other points in P in a particular order, say $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, based on the principle of the *regular tetrahedron* (to be elaborated in detail later) and compute the relative-distance representation according to all the 6 pairwise distances among these 4 points. The relative-distance representation of point $p_{(1)}$ wrt the other 3 points (i.e., $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$), denoted by $rd(p_{(1)}|p_{(2)},p_{(3)},p_{(4)})$, is defined as follows. That is, $rd(p_{(1)}|p_{(2)},p_{(3)},p_{(4)}) =$

$$(\|p_{(1)},p_{(2)}\|, \|p_{(1)},p_{(3)}\|, \|p_{(1)},p_{(4)}\|, \|p_{(2)},p_{(3)}\|, \|p_{(2)},p_{(4)}\|, \|p_{(3)},p_{(4)}\|) \quad (4)$$

We define the *first, second, third, fourth owners* of this representation to be $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, respectively. The donut representation of a point cloud is defined to be a collection of relative-distance representations of all points in

Notation	Description
$\mathscr{P}$	A database
n	The size of a database
Q	The query point cloud
P	A point cloud in the database
Θ	A rigid transformation
$T_\Theta(p)$	The transformed point of point p wrt Θ
$T_\Theta(P)$	The transformed point cloud of P wrt Θ
$corr(q,P)$	The correspondence point of point q on P
$\|q,p\|$	The Euclidean distance between point q and point p
$dist(Q,P)$	The distance between Q and P in different coordinate systems
$rd(p_{(1)}\|p_{(2)},p_{(3)},p_{(4)})$	The relative-distance representation of point $p_{(1)}$ wrt $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$
$ball(p,r)$	The r-sized ball of p
$ball\text{-}NN(p,r\|P)$	The nearest neighbor of the r-sized ball of p in P
$donut(p,r,\mathbf{n})$	The r-sized donut of p with its normal $\mathbf{n}$
$donut\text{-}NN(p,r,\mathbf{n}\|P)$	The nearest neighbor of the r-sized donut of p with its normal $\mathbf{n}$ in P
$sbz(p,[r_1,r_2])$	The sphere boundary zone of p with interval $[r_1,r_2]$
$dbz(D,[r_1,r_2])$	The donut boundary zone of D with interval $[r_1,r_2]$

Table 1 Commonly Used Notations

this point cloud. More importantly, since due to the property that the relative-distance representation considers only *relative* distance computation, as described in Section 1, this donut representation (or the relative-distance representation) satisfies the translation-invariant property and the rotation-invariant property. These properties ensure that given a set S of 4 points, even if we obtain a set S' of 4 points which are the 4 resulting points of S rotated with any angle and translated with any value where each of these points may deviate from its original coordinate with some distance, the relative-distance representation generated based on S is "close" to the relative-distance representation generated based on S'.

After we have the concept of "relative-distance representation", one may give the following naive implementation of framework C_2O when we consider a query point cloud Q and *one* data point cloud P. In Step 1, given a query point cloud Q, we randomly pick a point q in Q to generate its *relative-distance representation* R_q. Based on the representation R_q, for each point p in a data point cloud P, we check with the relative-distance representation of p, say R_p. If R_q and R_p are close, we can find the (coarse) transformation Θ according to the 4 selected points in Q and the 4 selected points in P. In Step 2, according to the (coarse) transformation Θ, we derive a (complete) transformation Θ_o based on all points in Q and all points in P using the state-of-the-art algorithm called Go-ICP [58] guaranteed to return the optimal transformation. In Step 3, we check whether $dist_{diff}(Q,P|\Theta_o) \leq \delta$. If the answer is yes, we include P in the output.

Since typically, there is more than one point cloud in the database, the naive implementation has to process one point cloud by one point cloud in the database and compare each point cloud in the database with the query point cloud, which is very time-consuming.

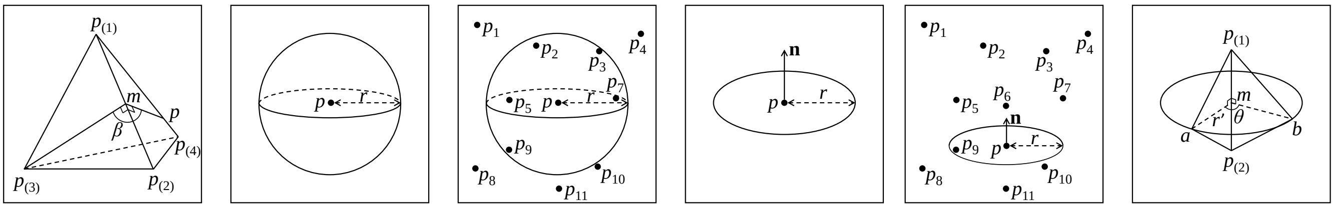
In this paper, we propose an index-based approach which involves 2 phases, namely the preprocessing phase and the query phase. In the preprocessing phase, we build an index on all point clouds in $\mathscr{P}$ according to the concept of "relative-distance representation". In the query phase, given a query point cloud, we select one point in Q and use its relative-distance representation to find the similar representations stored in the index efficiently.

4 Algorithm C_2O

We first give the details of the donut representation (containing a number of relative-distance representations of points in a point cloud) in Section 4.1. Then, we describe how the relative-distance representation of a point in a *data* point cloud could be matched with that of a point in a *query* point cloud in Section 4.2. Finally, we give the details of our proposed algorithm C_2O in Section 4.3.

4.1 Donut Representation

In this section, we give the details of the donut representation. The major principle of this donut representation is to generate a 6-dimensional tuple (called the *relative-distance representation*) according to the concept of "regular tetrahedron". There are two reasons why we use the concept of "regular tetrahedron". The first reason is due to the usage of a tetrahedron involving 4 points. It is found in [7] that using 4 points to represent some local structures of a point cloud could give a more differentiating power for pruning [7] (compared with using 3 points which is another branch in the literature [15]). The second reason is due to the usage of "regularity". It is found in [40] that more regular shapes could have more differentiating power [40]. Note that our proposed concept of "regular tetrahedron" is different from existing 4-point representations [7,40,41] in the



(a) (b) (c) (d) (e) (f)

Fig. 5 Examples for (a) Tetrahedron, (b) $ball(p,r)$, (c) $ball\text{-}NN(p,r|P)$, (d) $donut(p,r,\mathbf{n})$, (e) $donut\text{-}NN(p,r,\mathbf{n}|P)$ and Regular Tetrahedron Formation

literature. Firstly, existing representations of 4 points are separated into two point-pairs (each connected with a line segment) and a connector connecting the two line segments, which leads to the costly two-step pruning that is first based on the two point-pairs and then based on the connector. We consider a "holistic" shape of 4 points (i.e., a tetrahedron), and thus our pruning can be efficiently done in one step. Secondly, the 4 points in our representation are *ordered* but the existing 4-point representations are *unordered*, so we avoid enumerating all 4-point combinations, required by existing representations, to search the corresponding 4 points in a database object given a 4-point representation in a query object. Thus, our representation is much more efficient during the search.

Before we describe the relative-distance representation, let us define "regular tetrahedron". Given 4 points in a 3D space, namely $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, a *tetrahedron* is defined to be a polyhedron composed of 4 triangular faces, 6 straight edges and 4 vertex corners where $p_{(1)}, p_{(2)}, p_{(3)}$ and $p_{(4)}$ form the 4 corners. We also say that this tetrahedron is represented by these 4 points. One example of a tetrahedron is shown in Figure 5(a). Note that m is a point on edge $p_{(1)}p_{(2)}$. Suppose that line $p_{(3)}m$ is perpendicular to line $p_{(1)}p_{(2)}$, and line pm is perpendicular to line $p_{(1)}p_{(2)}$. Clearly, $\beta = \angle p_{(3)}mp$ is the dihedral angle between face $p_{(1)}p_{(2)}p_{(3)}$ and face $p_{(1)}p_{(2)}p_{(4)}$. The *size* of a tetrahedron is defined to be the maximum length of an edge in the tetrahedron. A tetrahedron is said to be *regular* if the lengths of edges are equal. Given a non-negative real number r, a tetrahedron is said to be an *r-sized regular tetrahedron* if the size of this tetrahedron is r and this tetrahedron is regular. Besides, the dihedral angle between any two adjacent faces in a regular tetrahedron is equal to $\arccos 1/3$ radian. Besides, it is easy to verify that in Figure 5(a), if the tetrahedron is regular, and m is the mid-point between $p_{(1)}$ and $p_{(2)}$, then the length of line $p_{(3)}m$ is equal to $\frac{\sqrt{3}}{2}\|p_{(1)}, p_{(2)}\|$. In this case, p is exactly $p_{(4)}$. Then, the length of line $p_{(4)}m$ (or line pm) is equal to $\frac{\sqrt{3}}{2}\|p_{(1)}, p_{(2)}\|$ too.

We are now ready to describe the relative-distance representation. Specifically, given a data point cloud P, for each point $p_{(1)}$ in P, we construct its relative-distance representation with the following steps.

- **Step 1 (Forming Tetrahedron):** We find 3 other points in P from $p_{(1)}$ in a *particular* order, namely $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, such that the 4 points found form a tetrahedron (where the 4 vertices of this tetrahedron are these 4 points) and this tetrahedron is as close as an R-sized regular tetrahedron where R is a non-negative real number which is roughly equal to a user parameter r, a non-negative real number.
- **Step 2 (Constructing Relative-Distance Representation):** According to these 4 points found, we construct the relative-distance representation (which is a 6-dimensional tuple) based on all the pairwise distances among these 4 points.

Step 2 is straightforward. Next, we give the details of Step 1. We are given a point $p_{(1)}$ in a point cloud P. Due to the unbalanced distribution of points in a point cloud, it is nearly impossible to construct an exact regular tetrahedron from $p_{(1)}$ and 3 other points in P. Thus, in the following, we describe how we find the 3 other points in P from $p_{(1)}$ in a particular order such that the constructed tetrahedron is "close" to a regular tetrahedron. Before that, we first define the concepts of "ball" and the concept of "donut".

Given a 3D point p and a non-negative real number r, the *r-sized ball of p*, denoted by $ball(p,r)$, is defined to be the surface of a sphere centered at p with radius equal to r. Given a 3D point p and a non-negative real number r, the *nearest neighbor of the r-sized ball of p* in P, denoted by $ball\text{-}NN(p,r|P)$, is defined to be the point in P which is the nearest to a point in the r-sized ball of p (i.e., $ball(p,r)$) and has its Euclidean distance to p at least r. Figure 5(b) shows an example of $ball(p,r)$ and Figure 5(c) shows that point p_{10} in P is $ball\text{-}NN(p,r|P)$.

Next, we define the concept of "donut". Given a 3D point p, a non-negative real number r and a 3D vector $\mathbf{n}$, the *r-sized donut of p with its normal* $\mathbf{n}$, denoted by $donut(p,r,\mathbf{n})$, is defined to be the boundary of a circle centered at p with radius equal to r which is on the plane with its (surface) normal equal to $\mathbf{n}$. Given a 3D point p, a non-negative real number r and a 3D vector $\mathbf{n}$, the *nearest neighbor of the r-sized donut of p with its normal* $\mathbf{n}$ in P, denoted by $donut\text{-}NN(p,r,\mathbf{n}|P)$, is defined to be the point in P which is nearest to a point in the r-sized donut of p with its normal $\mathbf{n}$ (i.e., $donut(p,r,\mathbf{n})$). Figure 5(d) shows an example of $donut(p,r,\mathbf{n})$ and Figure 5(e) shows that point p_9 in P is $donut\text{-}NN(p,r,\mathbf{n}|P)$.

We defined the concept of "ball" and the concept of "donut". Based on these 2 concepts, we can give our principle of constructing a tetrahedron involving 4 points. In order to construct a tetrahedron T involving 4 points from P (i.e., $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$) (which is not necessarily regular in most cases), we construct a *virtual* regular tetrahedron. In this virtual regular tetrahedron involving 4 points in a particular order, the first 2 points are points in P (which are assigned to $p_{(1)}$ and $p_{(2)}$, respectively), and the latter 2 points are *virtual* points, namely a and b (which may not be in P). The size of this regular tetrahedron (denoted by R) is determined in the following process but its value is near to a user parameter r (note that r can be easily chosen experimentally as we introduce in Section 6). In the final tetrahedron T involving 4 points (which come from P), the first 2 points are exactly $p_{(1)}$ and $p_{(2)}$, and the final 2 points, namely $p_{(3)}$ and $p_{(4)}$, are "close" to points a and b, respectively.

Algorithm 1 presents the steps of finding these 4 points in P (together with points a and b). We start with a point in P, say $p_{(1)}$. In the following steps, we find $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, respectively.

- **Finding Second Point $p_{(2)}$:** According to a user parameter r (which is a non-negative real number), we find another point in P, say $p_{(2)}$, which is the nearest to $p_{(1)}$ with distance from $p_{(1)}$ at least r (i.e., $ball\text{-}NN(p_{(1)},r|P)$). The step of finding $p_{(2)}$ is shown in Line 1 of Algorithm 1.
- **Finding Third Point $p_{(3)}$:** Based on $p_{(1)}$ and $p_{(2)}$, we form a virtual regular tetrahedron as follows where the size of this tetrahedron (i.e., R) is equal to $\|p_{(1)},p_{(2)}\|$ (typically, R is close to r in practice and in our experiments, R is at most 1.1 times r). We construct a *locus* of a point denoting D such that this point has its distance to $p_{(1)}$ and its distance to $p_{(2)}$ both equal to $\|p_{(1)},p_{(2)}\|$. Let m be the mid-point between $p_{(1)}$ and $p_{(2)}$, and $\mathbf{n}$ be a vector from $p_{(2)}$ to $p_{(1)}$. It is easy to verify that this locus is equal to $donut(m,r',\mathbf{n})$, where r' is defined to be $\frac{\sqrt{3}}{2}\|p_{(1)},p_{(2)}\|$. Since the shape of the locus is similar to "donut", this is the reason why we call this as the donut representation. According to this locus, we find the point in P nearest to this locus (i.e., $donut\text{-}NN(m,r',\mathbf{n}|P)$), which is assigned to $p_{(3)}$. The steps of finding $p_{(3)}$ are shown in Line 2–6 of Algorithm 1.
- **Finding Fourth Point $p_{(4)}$:** Then, the point on the locus D nearest to $p_{(3)}$ can be determined and is assigned to a. Based on $p_{(1)}$, $p_{(2)}$ and a, we can virtually construct a regular tetrahedron with 4 vertices. The first 3 vertices of the virtual regular tetrahedron are $p_{(1)}$, $p_{(2)}$ and a. It is easy to know that the last vertex of this virtual regular tetrahedron (i.e., b) can be found on two possible positions along the locus. For a more deterministic formation of the regular tetrahedron, we adopt the strategy to find the position (among these 2 positions) as the position of point b which is the position closest to a in the anti-clockwise direction from a on the locus (in the view point from $p_{(1)}$ to $p_{(2)}$). Figure 5(f) illustrates point b in a regular tetrahedron. Then, we find the point in P nearest to b, which is assigned to $p_{(4)}$. The steps of finding $p_{(4)}$ (together with points a and b) are shown in Line 7–9 of Algorithm 1.

Algorithm 1 Forming Tetrahedron

Input: Point cloud P, point $p_{(1)}$ in P, non-negative real number r
Output: Point $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$
1: $p_{(2)} \leftarrow ball\text{-}NN(p_{(1)},r|P)$
2: $m \leftarrow$ the mid-point between $p_{(1)}$ and $p_{(2)}$
3: $\mathbf{n} \leftarrow$ a vector from $p_{(2)}$ to $p_{(1)}$
4: $r' \leftarrow \frac{\sqrt{3}}{2}\|p_{(1)},p_{(2)}\|$
5: $D \leftarrow donut(m,r',\mathbf{n})$
6: $p_{(3)} \leftarrow donut\text{-}NN(m,r',\mathbf{n}|P)$
7: $a \leftarrow$ the point on D nearest to $p_{(3)}$
8: $b \leftarrow$ the point on the position closest to a in the anti-clockwise direction from a on D (in the view point from $p_{(1)}$ to $p_{(2)}$) with an angle of $\arccos(1/3)$ radian
9: $p_{(4)} \leftarrow$ the point in P nearest to b
10: **return** $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$

Finally, we obtain all 4 points in P (i.e., $p_{(1)},p_{(2)},p_{(3)}$ and $p_{(4)}$).

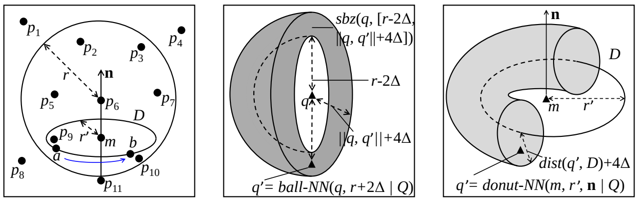
Consider our example as shown in Figure 6(a). Suppose we want to form a tetrahedron starting from point p_6. Note that $p_{(1)} = p_6$. We first assign p_{11} as $p_{(2)}$, since $p_{11} = ball\text{-}NN(p_6,r|P)$. Thus, $p_{(2)} = p_{11}$. Secondly, let m be the mid-point between p_6 and p_{11}. Let $\mathbf{n}$ be a vector from p_{11} to p_6. Let $r' = \frac{\sqrt{3}}{2}\|p_{(1)},p_{(2)}\|$. We denote $donut(m,r',\mathbf{n})$ by D. It can be seen that $donut\text{-}NN(m,r',\mathbf{n}|P)$ is equal to p_9, which is thus assigned to $p_{(3)}$. Note that a is the nearest point to p_9 on donut D. Thirdly, in the figure, b is a point on D which is in the anti-clockwise direction from a (indicated by the blue arrow) (in the view point from p_6 to p_{11}). We find the nearest point in P to b (i.e., p_{10}) which is assigned to $p_{(4)}$.

Note that the above steps generate a tetrahedron (represented by $p_{(1)}$, $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$) which is close to a regular tetrahedron (represented by $p_{(1)}$, $p_{(2)}$, a and b). With these 4 points (i.e., $p_{(1)},p_{(2)},p_{(3)}$ and $p_{(4)}$), we can construct the relative-distance representation as shown in Equation 4.

4.2 Matching Relative-Distance Representation

In this section, we describe how the relative-distance representation of a point in a *data* point cloud can be matched with that of a point in a *query* point cloud even though there is an order enforced when we find the 3 other points from a point (e.g., $p_{(1)}$).

Consider a query point cloud Q and a data point cloud P in $\mathscr{P}$. In our problem, we want to determine whether



(a) (b) (c)

Fig. 6 Examples for (a) Tetrahedron Formation, (b) Sphere Bounding Zone and (c) Donut Bounding Zone

$dist(Q,P) \le \delta$. When δ is greater than 0, this means that after an optimal transformation on Q resulting in a transformed point cloud Q', each point q' in Q' has its correspondence point p on P such that $\|q',p\|$ *could* be greater than 0. However, since δ is a fixed value, $\|q',p\|$ can be upper bounded by a fixed value too. The following lemma shows the upper bound of $\|q',p\|$ according to δ.

Lemma 1 *Let Q' be the point cloud optimally transformed from Q (wrt P). If $dist(Q,P) \le \delta$, then for each point q' in Q' and its correspondence point p on P, $\|q',p\| \le \delta|Q|^{1/2}$.*

Proof Applying Equation 3 where $Q' = T_{\Theta_o}(Q)$, we have $dist(Q,P) = [\frac{1}{|Q|}\sum_{q\in Q'}\|q',corr(q',P)\|^2]^{1/2} \le \delta$, and it is trivially derived that $\sum_{q\in Q'}\|q',corr(q',P)\|^2 \le \delta^2|Q|$. Therefore,

$$\begin{aligned}\|q',p\| &= (\|q',p\|^2)^{1/2}\\ &\le [\sum_{q\in Q'}\|q',corr(q',P)\|^2]^{1/2}\\ &\le (\delta^2|Q|)^{1/2} = \delta|Q|^{1/2}\end{aligned}$$

□

Let $\triangle = \delta|Q|^{1/2}$. Based on this lemma, we derive a property called *Distance Bound Property* stating that the distance between each (transformed) query point and its correspondence on P is bounded by $\triangle$. Note that the bound $\triangle$ is tight since it is possible that $\|q',p\| = \triangle$.

We are ready to describe how the relative-distance representation of a point in the query point cloud Q (generated based on 4 points in Q) can be used to "map" with the relative-distance representation of a point in the data point cloud P (generated based on 4 points in P) in the following steps.

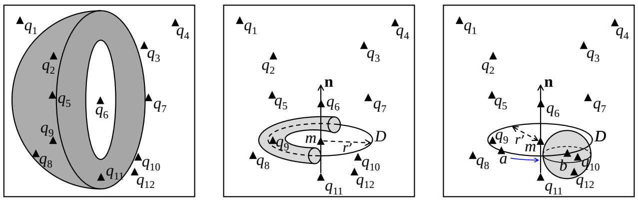
- **Step (a) (Constructing Relative-Distance Representation Candidates):** We construct a set of *candidates* of the relative-distance representation of a point in Q, say $q_{(1)}$.
- **Step (b) (Finding Answers from Database):** For each *candidate* c in Step (a), we find a list of point clouds in $\mathscr{P}$ according to the relative-distance representations of all point clouds in $\mathscr{P}$ and the candidate c.

We first assume that $q_{(1)}$ is an arbitrarily selected point in Q. Later, in Section 4.3.2, we introduce our strategy to find the point $q_{(1)}$ in Q which could lead to the optimal efficiency.

Step (a) is straightforward. Doing Step (b) *efficiently* is non-trivial. Consider a point cloud P in $\mathscr{P}$. Suppose that $dist(Q,P) \le \delta$. We know that each point q in Q has its correspondence point on P (i.e., $corr(q,P)$) when we compute $dist(Q,P)$. Since each point in P finds 3 other points in P in a *particular order* (based on the principle of regular tetrahedron) and constructs its relative-distance representation, for effective mapping between Q and P based on their relative-distance representations, one method of finding 3 other points in Q from a point $q_{(1)}$ (from Step (a)) whose correspondence point on P is $p_{(1)}$ is described as follows. Let $p_{(2)}, p_{(3)}$ and $p_{(4)}$ be the second, third and fourth owners of the relative-distance representation of $p_{(1)}$, respectively. The method is to find an ordering of the other 3 points in Q, say $q_{(2)}, q_{(3)}$ and $q_{(4)}$, whose correspondence points on P are $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$, respectively. In other words, the ordering of the 4 points in Q (i.e., $(q_{(1)}, q_{(2)}, q_{(3)}, q_{(4)})$) is consistent with the ordering of the 4 points in P (i.e., $(p_{(1)}, p_{(2)}, p_{(3)}, p_{(4)})$).

One naive but *inefficient* implementation of Step (a) is to enumerate *all combinations* in the Cartesian product of $\{q_{(1)}\} \times Q \times Q \times Q$, denoting all possible orderings of 4 points in Q starting from $q_{(1)}$, and to construct their relative-distance representations. Although one combination (denoting one ordering of 4 points in Q) is consistent with the ordering of their (correspondence) points on P, this implementation is quite expensive. In the following, we describe how to do it efficiently by finding a *small* subset of these combinations only. The major idea is to generate this small (candidate) set of these combinations based on two major principles. The first principle is to follow Distance Bound Property and the second principle is to follow how to find the 4 points in a point cloud (as described in Section 4.1). Once we obtain the candidate set in Step (a), we can do Step (b) easily by comparing each candidate in Step (a) with point clouds in $\mathscr{P}$ according to their relative-distance representations. Later, in Section 4.3, we describe how we use an index to further speed up this step.

In the following, we propose an efficient method to find a set of candidates much more *efficiently* for Step (b). Firstly, we start from point $q_{(1)}$ (obtained from Step (a)). Secondly, based on the two major principles, we find a set $S_{(2)}$ of candidate points in Q for point $q_{(2)}$ according to $q_{(1)}$ (Section 4.2.1). Similarly, based on the principles, we find a set $S_{(3)}$ of candidate points in Q for point $q_{(3)}$ according to $q_{(1)}$ and $q_{(2)}$ (Section 4.2.2). We do the same to find a set $S_{(4)}$ for point $q_{(4)}$ according to $q_{(1)}, q_{(2)}$ and $q_{(3)}$ (Section 4.2.3). Then, we construct a candidate set of all relative-distance representations according to $\{q_{(1)}\}$, $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ (Section 4.2.4). This candidate set corresponds to the output of



(a) Finding $S_{(2)}$ (b) Finding $S_{(3)}$ (c) Finding $S_{(3)}$

Fig. 7 Examples of Finding $S_{(2)}$, Finding $S_{(3)}$ and Finding $S_{(4)}$

Step (b). Note that $\{q_{(1)}\}$ is much smaller than Q and $S_{(i)}$ is also much smaller than Q for each $i \in [2,4]$, and thus, this candidate set is much smaller than the Cartesian product of $Q \times Q \times Q \times Q$.

4.2.1 Finding $S_{(2)}$

In the following, we describe how to find $S_{(2)}$ according to $q_{(1)}$ first based on the two principles. Before that, we first define some concepts whose major ideas come from the two principles. Given two non-negative real numbers, namely r_1 and r_2, where $r_1 \leq r_2$, we define an *interval*, denoted by $[r_1, r_2]$, to represent all values such that each value is at least r_1 and at most r_2. Given a 3D point p and an interval $[r_1, r_2]$ where $r_1 < r_2$, the *sphere boundary zone of p with interval* $[r_1, r_2]$, denoted by $sbz(p, [r_1, r_2])$, is defined to be the 3D space containing all 3D points such that each point in this space has its distance to p at least r_1 and at most r_2.

In Figure 6(b), the shaded region is $sbz(q, [r_1, r_2])$ where $r_1 = r - 2\triangle$, $r_2 = \|q, q'\| + 4\triangle$ and $q' = ball\text{-}NN(q, r + 2\triangle|Q)$. For better illustration, in the figure, we show $sbz(q, [r_1, r_2])$ and similar concepts in the "half-sphere", and the other "half-sphere" is just symmetric. Note that it is obvious $r_1 = r - 2\triangle \leq r \leq r_2 = \|q, q'\| + 4\triangle$.

Next, we give the following lemma based on the concepts of the sphere bounding zone, which can help to construct $S_{(2)}$. The idea is that we would like to find a small subset of Q, denoted by $S_{(2)}$, which must contain the desired $q_{(2)}$ having $p_{(2)}$ as the correspondence point on P. Since $p_{(2)}$ is the nearest neighbor of the r-sized ball of $p_{(1)}$ in P (and is very close to the r-sized ball of $p_{(1)}$ in most cases), based on the two major principles, it is guaranteed that our desired $q_{(2)}$ is also close to the r-sized ball of $q_{(1)}$ or the distance from our desired $q_{(2)}$ to $q_{(1)}$ is close to the distance from $p_{(2)}$ to $p_{(1)}$.

Lemma 2 *Consider a query point cloud Q and a database point cloud P. Let* $q_{(1)}$ *be a point in Q whose correspondence point on P is* $p_{(1)}$*. Let* $p_{(2)}, p_{(3)}$ *and* $p_{(4)}$ *be the second, third and fourth owners of the relative-distance representation of* $p_{(1)}$*, respectively. Let* $S_{(2)}$ *be the set of all points from Q in* $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$ *where* $q' = ball\text{-}NN(q_{(1)}, r + 2\triangle|Q)$*. If* $dist(Q,P) \leq \delta$*, then there exists a point* $q_{(2)}$ *in* $S_{(2)}$ *such that* $p_{(2)}$ *is the correspondence point of* $q_{(2)}$ *on P.*

Proof Firstly, we have the following claim, which can be easily derived from Lemma 1 and will be useful in our proof.

Claim Consider a query point cloud Q and a database point cloud P. Let q and q' be two points in Q whose correspondence point in P are respectively p and p'. If $dist(Q,P) \leq \delta$, then we have

$$\|p, p'\| - 2\triangle \leq \|q, q'\| \leq \|p, p'\| + 2\triangle$$

$$\|q, q'\| - 2\triangle \leq \|p, p'\| \leq \|q, q'\| + 2\triangle$$

Also, according to Lemma 1, since $dist(Q,P) \leq \delta$, there exists $q_{(2)} \in Q$, such that $\|q_{(1)}, p_{(1)}\| \leq \triangle$ and $\|q_{(2)}, p_{(2)}\| \leq \triangle$. Now, we show that $q_{(2)}$ is in $S_{(2)}$, which is the set of all points from Q in $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$ where $q' = ball\text{-}NN(q_{(1)}, r + 2\triangle|Q)$. By definition, we need to show that $r - 2\triangle \leq \|q_{(1)}, q_{(2)}\| \leq \|q_{(1)}, q'\| + 4\triangle$.

Since $q' = ball\text{-}NN(q_{(1)}, r + 2\triangle|Q)$, we have $\|q_{(1)}, q'\| \geq r + 2\triangle \geq r - 2\triangle$. Thus, we show that none of the following cases is possible: $\|q_{(1)}, q_{(2)}\| < r - 2\triangle$ or $\|q_{(1)}, q_{(2)}\| > \|q_{(1)}, q'\| + 4\triangle$.

(1) Since $p_{(2)} = ball\text{-}NN(p_{(1)}, r|P)$, we have $\|p_{(1)}, p_{(2)}\| \geq r$. Therefore, by the above claim, $\|q_{(1)}, q_{(2)}\| \geq \|p_{(1)}, p_{(2)}\| - 2\triangle \geq r - 2\triangle$, indicating that the first inequality cannot hold.

(2) Assuming the third inequality holds, since $\|q_{(1)}, q_{(2)}\| \leq \|p_{(1)}, p_{(2)}\| + 2\triangle$ (by the above claim), we have $\|p_{(1)}, p_{(2)}\| > \|q_{(1)}, q'\| + 2\triangle$. Let p' be the correspondence point to q' in P. Again by the above claim, we have $\|p_{(1)}, p'\| \in [\|q_{(1)}, q'\| - 2\triangle, \|q_{(1)}, q'\| + 2\triangle]$, and thus we must have $\|p_{(1)}, p'\| < \|p_{(1)}, p_{(2)}\|$. Moreover, since $\|q_{(1)}, q'\| \geq r + 2\triangle$, the lower bound of $\|p_{(1)}, p'\|$ is r, indicating that point p' is closer to the r-sized ball centered at $p_{(1)}$. This leads to a contradiction with $p_{(2)} = ball\text{-}NN(p_{(1)}, r|P)$.

Therefore, we have $q_{(2)} \in sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$. Since $q_{(2)}$ is a point of Q, by definition, we have $q_{(2)} \in S_{(2)}$. □

If there exists a point cloud P such that $dist(Q,P) \leq \delta$ (which means that each point in Q (including point $q_{(2)}$) has its correspondence point in P), according to the above lemma, we know that, given $q_{(1)}$ (whose correspondence point on P is $p_{(1)}$), we can find a set $S_{(2)}$ of *candidate* points in Q for $q_{(2)}$, such that $S_{(2)}$ must contain $q_{(2)}$ which has its correspondence point on P as $p_{(2)}$, where $p_{(2)}$ is the second owner of the relative-distance representation of $p_{(1)}$ and can be listed out.

Figure 7(a) shows an example of finding $S_{(2)}$ in Q if we pick q_6 as $q_{(1)}$. The shaded region (which is also shown in the "half-sphere" for better illustration and the other "half-sphere" is symmetric) represents $sbz(q_6, [r - 2\triangle, \|q_6, q'\| +$

$4\triangle])$ where q_{11} is selected to be q' since $q_{11} = ball\text{-}NN(q_6, r+2\triangle|Q)$. Thus, $S_{(2)}$ is the set of all the points inside the shaded region (i.e., $S_{(2)} = \{q_2, q_8, q_9, q_{11}\}$). Note that q_1, q_3, q_4, q_5, q_6, q_7, q_{10} and q_{12} in the figure are outside the shaded region.

4.2.2 Finding $S_{(3)}$

Next, we consider how to construct $S_{(3)}$ according to $q_{(1)}$ and $q_{(2)}$ based on the two principles where $q_{(2)} \in S_{(2)}$. Before that, we also define some concepts. Given a point q' and a donut D, the distance between q' and D, denoted by $dist(q', D)$, is defined to be the distance between q' and its nearest point on D. Given a donut D and an interval $[r_1, r_2]$, we define the emphdonut boundary zone of D with an interval $[r_1, r_2]$, denoted by $dbz(D, [r_1, r_2])$, to be the 3D space containing all 3D points such that each point in this space has distance to donut D at least r_1 and at most r_2. In Figure 6(c), the shaded region is $dbz(D, [0, dist(q', D) + 4\triangle])$ (shown in "half-space").

Based on the concept of donut boundary zone, we show the following lemma for constructing $S_{(3)}$ that contains the desired $q_{(3)}$ when we are given $q_{(1)}$ and $q_{(2)}$ (whose correspondence points on P are $p_{(1)}$ and $p_{(2)}$, respectively). The major ideas are as follows. In Section 4.1, for a *database* point cloud P, the third owner of the relative-distance representation of $p_{(1)}$ (i.e., $p_{(3)}$) is the nearest point in P to a donut (constructed from $p_{(1)}$ and $p_{(2)}$ in P). Thus, the desired $q_{(3)}$ with $p_{(3)}$ as its correspondence point on P is also close to a donut constructed similarly from $q_{(1)}$ and $q_{(2)}$, due to the property that the distance from each query point to its correspondence point can be bounded (by Lemma 1).

Lemma 3 *Consider a query point cloud Q and a database point cloud P. Let $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$ be the second, third and fourth owners of the relative-distance representation of point $p_{(1)}$ in P, respectively. Let $q_{(1)}$ and $q_{(2)}$ be the points in Q whose correspondence point on P are $p_{(1)}$ and $p_{(2)}$, respectively. Let m be the mid-point between $q_{(1)}$ and $q_{(2)}$, $r' = \frac{\sqrt{3}}{2}\|q_{(1)}, q_{(2)}\|$, and $\mathbf{n}$ be the vector from $q_{(2)}$ to $q_{(1)}$. Let $D = donut(m, r', \mathbf{n})$ and $q' = donut\text{-}NN(m, r', \mathbf{n}|Q)$. Let $S_{(3)}$ be the set of all points from Q in $dbz(D, [0, dist(q', D) + 4\triangle])$. If $dist(Q, P) \leq \delta$, then there exists a point $q_{(3)}$ in $S_{(3)}$ such that $p_{(3)}$ is the correspondence point of $q_{(3)}$ on P.*

Proof Since $dist(Q, P) \leq \delta$, by Lemma 1, $q_{(3)}$ exists in Q. Since $S_{(3)}$ is defined to be the set of all points in Q whose distance to the donut D is at least $dist(q', D)$ and at most $dist(q', D) + 4\triangle$ where q' is defined to be the nearest point to D, we assume that $q_{(3)}$ does not suffice this condition. That is, $dist(q_{(3)}, D) < dist(q', D)$ or $dist(q_{(3)}, D) > dist(q', D) + 4\triangle$. If the former holds, it contradicts the definition of q', indicating that $q_{(3)}$ instead of q' should be the nearest point to D in this case. If the latter holds, according to the claim in the proof of Lemma 2, we have $dist(p_{(3)}, D) \geq dist(q_{(3)}, D) - 2\triangle$ and $dist(q', D) + 2\triangle \geq dist(p', D)$, where p' is the correspondence point of q' in P. Then, $dist(p_{(3)}, D) \geq dist(q_{(3)}, D) - 2\triangle > dist(q', D) + 4\triangle - 2\triangle = dist(q', D) + 2\triangle \geq dist(p', D)$. This indicates that p' has smaller distance to donut D, which leads to a contradiction since $p_{(3)}$ is defined to be the nearest point to D. Therefore, $q_{(3)}$ must be found and included in $S_{(3)}$. □

According to the above lemma, given $q_{(1)}$ and $q_{(2)}$ (whose correspondence points on P are $p_{(1)}$ and $p_{(2)}$, respectively), we can also find a set $S_{(3)}$ of *candidate* points in Q for $q_{(3)}$ inside region $dbz(D, [0, dist(q', D) + 4\triangle])$ such that $S_{(3)}$ must contain $q_{(3)}$ which has its correspondence point on P as $p_{(3)}$, where $p_{(3)}$ is the third owner of the relative-distance representation of $p_{(1)}$ and can be listed out.

Consider the example shown in Figure 7(b) where we pick q_6 as $q_{(1)}$ and q_{11} as $q_{(2)}$. Accordingly, we find $D = donut(m, r', \mathbf{n})$ where m is the mid-point between q_6 and q_{11}, $r' = \frac{\sqrt{3}}{2}\|q_6, q_{11}\|$ and $\mathbf{n}$ is the vector from q_{11} to q_6. The shaded region (shown in the "half-space") represents $dbz(D, [0, dist(q', D) + 4\triangle])$ where q_9 is selected to be q' since $q_9 = donut\text{-}NN(m, r', \mathbf{n}|Q)$. Thus, $S_{(3)}$ is a set containing q_9 and q_{10} since they are inside the shaded region.

4.2.3 Finding $S_{(4)}$

We also consider how to construct $S_{(4)}$ according to $q_{(1)}$, $q_{(2)}$ and $q_{(3)}$ based on the two principles where $q_{(2)} \in S_{(2)}$ and $q_{(3)} \in S_{(3)}$. Since the fourth owner of the relative-distance representation of $p_{(1)}$ (i.e., $p_{(4)}$) is close to the second virtual point in the steps in Section 4.1, we can find the desired $q_{(4)}$ with $p_{(4)}$ as its correspondence point on P near the second virtual point in Q constructed in the same way. The following lemma presents how we construct $S_{(4)}$ that contains the desired $q_{(4)}$.

Lemma 4 *Consider a query point cloud Q and a database point cloud P. Let $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$ be the second, third, fourth owners of the relative-distance representation of point $p_{(1)}$ in P, respectively. Let $q_{(1)}$, $q_{(2)}$ and $q_{(3)}$ be the points in Q whose correspondence point on P are $p_{(1)}$, $p_{(2)}$ and $p_{(3)}$, respectively. Let m be the mid-point between $q_{(1)}$ and $q_{(2)}$, $r' = \frac{\sqrt{3}}{2}\|q_{(1)}, q_{(2)}\|$, and $\mathbf{n}$ be the vector from $q_{(2)}$ to $q_{(1)}$. Let $D = donut(m, r', \mathbf{n})$. Let a be a point on D that is nearest to $q_{(3)}$. Let b be the point at the position nearest to a in the anti-clockwise direction from a on D in the view point from $q_{(1)}$ to $q_{(2)}$ (such that $q_{(1)}$, $q_{(2)}$, a and b form a regular tetrahedron). Let q' be the nearest neighbor of b in Q. Let $S_{(4)}$ be the set of all points from Q in $sbz(b, [0, \|b, q'\| + 4\triangle])$. If $dist(Q, P) \leq \delta$, then there exists a point $q_{(4)}$ in $S_{(4)}$ such that $p_{(4)}$ is the correspondence point of $q_{(4)}$ on P.*

Proof Since $dist(Q,P) \leq \delta$, by Lemma 1, $q_{(4)}$ also exists in Q. Since $S_{(4)}$ is defined to be the set of all points in Q inside $R_{NN} = sbz(b, [\|b,q'\|, \|b,q'\| + 4\triangle])$ whose distance to point b is at least $\|b,q'\|$ and at most $\|b,q'\| + 4\triangle$ where q' is defined to be the nearest neighbor of b, we assume that $q_{(4)}$ does not suffice this condition. That is, $\|b,q_{(4)}\| < \|b,q'\|$ or $\|b,q_{(4)}\| > \|b,q'\| + 4\triangle$. If the former holds, it contradicts the definition of q', indicating that $q_{(4)}$ instead of q' should be the nearest neighbor of b in this case. If the latter holds, according to the claim in the proof of Lemma 2, we have $\|b,p_{(4)}\| \geq \|b,q_{(4)}\| - 2\triangle$ and $\|b,q'\| + 2\triangle \geq \|b,p'\|$, where p' is the correspondence point of q' in P. Then, $\|b,p_{(4)}\| \geq \|b,q_{(4)}\| - 2\triangle > \|b,q'\| + 4\triangle - 2\triangle = \|b,q'\| + 2\triangle \geq \|b,p'\|$. This indicates that p' has smaller distance to b, which leads to a contradiction since $p_{(4)}$ is defined to be the nearest neighbor of b. Therefore, $q_{(4)}$ must be found and included in $S_{(4)}$. □

Based on the above lemma, given $q_{(1)}$, $q_{(2)}$ and $q_{(3)}$ (whose correspondence points on P are $p_{(1)}$, $p_{(2)}$ and $p_{(3)}$, respectively), we can find a set $S_{(4)}$ of *candidate* points in Q for $q_{(4)}$ inside region $sbz(b, [0, \|b,q'\| + 4\triangle])$, such that $S_{(4)}$ must contain $q_{(4)}$ which has its correspondence point on P as $p_{(4)}$, where $p_{(4)}$ is the fourth owner of the relative-distance representation of $p_{(1)}$ and can be listed out.

In the example shown in Figure 7(c), we pick q_6 as $q_{(1)}$, q_{11} as $q_{(2)}$ and q_9 as $q_{(3)}$. Point a is the nearest point of q_9 on D, and point b is a point on D which is in the anti-clockwise direction from a (indicated by the blue arrow) (in the view point from q_6 to q_{11}). We find q_{10} to be the nearest neighbor of b in Q (i.e., $q'' = q_{10}$), and thus we find $sbz(b, [0, \|b,q''\| + 4\triangle])$ shown as the shaded region in the figure. Since the shaded region also covers q_{12}, $S_{(4)} = \{q_{10}, q_{12}\}$.

4.2.4 Constructing Candidate Set

Now, we present how to construct the candidate set of relative-distance representations according to $\{q_{(1)}\}$, $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ (i.e., the details of Step (a)). We first initialize a variable $\mathscr{C}$, to store the results of the candidates (in the form of a set of relative-distance representations), to an empty set.

- **Step (i) (Finding $S_{(2)}$):** Firstly, we find point q' to be *ball-NN*$(q_{(1)}, r + 2\triangle | Q)$. Secondly, we find set $S_{(2)}$ containing all points in Q inside $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$. Thirdly, for each point $q_{(2)}$ in $S_{(2)}$, we do the following step.
  - **Step (i)(1) (Finding $S_{(3)}$):** Firstly, we find point m to be the mid-point between $q_{(1)}$ and $q_{(2)}$, we find vector $\mathbf{n}$ to be the vector from $q_{(2)}$ to $q_{(1)}$ and we find r' to be $\frac{\sqrt{3}}{2}\|q_{(1)}, q_{(2)}\|$. Secondly, we find donut D to be *donut*$(m, r', \mathbf{n})$. Thirdly, we find point q' to be *donut-NN*$(m, r', \mathbf{n} | Q)$. Fourthly, we find set $S_{(2)}$ containing all points in Q inside $dbz(D, [0, dist(q', D) + 4\triangle])$. Fifthly, for each point $q_{(3)}$ in $S_{(3)}$, we do the following step.
    - **Step (i)(1)(I) (Finding $S_{(4)}$):** Firstly, we find point a to be a point on D that is nearest to $q_{(3)}$. Secondly, we find point b to be the point at the position nearest to a in the anti-clockwise direction from a on D (in the view point from $q_{(1)}$ to $q_{(2)}$). Thirdly, we find point q'' to be the nearest neighbor of point b in Q. Fourthly, we find set $S_{(4)}$ containing all points in Q inside $sbz(b, [0\ \|b,q''\| + 4\triangle])$. Fifthly, for each point $q_{(4)}$ in $S_{(4)}$, we do the following step.
      - **Step (i)(1)(I)-1 (Constructing all relative-distance representations):** Firstly, we obtain the relative-distance representation R to be $rd(q_{(1)} | q_{(2)}, q_{(3)}, q_{(4)})$ by Equation 4. Secondly, we associate R with $q_{(1)}$, $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$. Thirdly, we insert R into the result set $\mathscr{C}$ of candidate relative-distance representations.

The following lemma shows the correctness of the above details for Step (a). Specifically, given a query point cloud Q and a database point cloud P, if $dist(Q,P) \leq \delta$, the output candidate set generated by Step (a) contains a relative-distance representation of an arbitrary point $q_{(1)}$ in Q, which has a *correspondence* relative-distance representation of a point on P, says $p_{(1)}$.

Lemma 5 *Consider a query point cloud Q and a database point cloud P. Let $q_{(1)}$ be a point in Q whose correspondence point on P is $p_{(1)}$. Let $p_{(2)}, p_{(3)}$ and $p_{(4)}$ be the second, third and fourth owners of the relative-distance representation of $p_{(1)}$, respectively. Let $\mathscr{C}_{q_{(1)}}$ be the output set of candidate relative-distance representations on Q with the starting point $q_{(1)}$. If $dist(Q,P) \leq \delta$, then there exist three points $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$ in Q, such that $p_{(i)}$ is the correspondence point of $q_{(i)}$ on P for $i \in [2,4]$, and $rd(q_{(1)} | q_{(2)}, q_{(3)}, q_{(4)}) \in \mathscr{C}_{q_{(1)}}$.*

Proof According to Lemma 1, since $dist(Q,P) \leq \delta$, there exists $q_{(i)} \in Q$ for all $1 \leq i \leq 4$, such that $\|q_{(i)}, p_{(i)}\| \leq \triangle$. By Lemma 2, $S_{(2)}$ must contain $q_{(2)}$. By Lemma 3, $S_{(3)}$ must contain $q_{(3)}$. By Lemma 4, $S_{(4)}$ must contain $q_{(4)}$. Therefore, the candidate relative-distance representation $rd(q_{(1)} | q_{(2)}, q_{(3)}, q_{(4)})$ will be included in the result set $\mathscr{C}_{q_{(1)}}$. □

Note that the size of the resulting candidate set (i.e., $|\mathscr{C}|$) could be $O(|Q|^3)$, since each one of $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ could be as large as $O(|Q|)$ in the worst case. However, our effective pruning strategies to find $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ ensure that $|\mathscr{C}|$ is generally small. In our experiments of typical settings (i.e., $|Q| = 100$), the number of candidate representations is only around 400, which is significantly smaller than $|Q|^3 = 10^6$. Later, in Section 4.3.2, we give our strategy choosing $q_{(1)}$ to further keep $|\mathscr{C}|$ as small as possible.

4.3 Two Phases of C_2O

We present the preprocessing phase and the query phase of C_2O in Section 4.3.1 and Section 4.3.2, respectively.

4.3.1 Preprocessing Phase

The preprocessing phase is straightforward with our donut representation. It involves the following steps. For each point cloud P in $\mathscr{P}$, we do the following sub-steps. The first sub-step is to build a 3D index I_P (e.g., R*-tree[1] [9]) on all points in P. The second sub-step is to perform the following procedure for each point $p_{(1)}$ in P. We construct the relative-distance representation of $p_{(1)}$ (i.e., $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ where $p_{(2)}$, $p_{(3)}$ and $p_{(4)}$ are the other 3 points found (as described before) with the help of index I_P. We insert $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ into a 6-dimensional index I_{DB} (e.g., R*-tree), initialized to an empty index. Note that in I_{DB}, each $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ is associated with two parts: (1) the ID of point cloud P in $\mathscr{P}$ and (2) the point IDs of the 4 points used to construct $rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$.

Clearly, the final index size of I_{DB} is $O(n)$ where n is the sum of the sizes of all database point clouds. Next, we discuss the preprocessing time. Firstly, the time complexity of building an index I_P of one point cloud P is $O(|P|\log|P|)$, and thus the total time complexity of building indices of all point clouds in the database $\mathscr{P}$ is $O(\sum_{P\in\mathscr{P}}|P|\log|P|) = O((\sum_{P\in\mathscr{P}}|P|)\log n) = O(n\log n)$, since each $|P|$ is at most n. Secondly, for each point in a database point cloud P, we obtain the relative-distance representation in expected $O(\log|P|)$ time with the help of I_P. Since there are n database points, and $|P|$ is at most n, obtaining all relative-distance representations takes $O(n\log n)$ time. The overall preprocessing time complexity is $O(n\log n)$.

Note that C_2O is generic for any other type of efficient multi-dimensional index (e.g., k-d tree [10]) to implement I_{DB}. In our experiments, we tested different index types, and R*-tree gives the best query efficiency.

4.3.2 Query Phase

We describe the following 2 steps of our query phase as follows. Consider a query point cloud Q.

Step 1 (Relative-Distance Representation Candidate Generation): Since the number of the generated candidates is crucial, to keep this number as small as possible, we use a simple and effective strategy as follows. Firstly, for each point in Q, we perform the steps described in Section 4.2 and obtain a set of candidate relative-distance representations of this point. Secondly, we select the point in Q such that its candidate set has the smallest size among all the generated candidate sets, and assign this point to $q_{(1)}$. Let $C_{q_{(1)}}$ denote the set of candidate relative-distance representations of $q_{(1)}$.

Step 2 (Point Cloud Matching): We first initialize a variable $\mathscr{R}$, which is to store a set of IDs of the point clouds in P as the query result, to an empty set. Then, we do the following steps.

- *Step a:* We initialize a variable $\mathscr{C}$, which is to store a set of 2-tuples each in the form of (R_Q, R_P) where R_Q (R_P) is a relative-distance representation of a point in Q (P), to an empty set.
- *Step b:* For each representation R_Q in $C_{q_{(1)}}$, we perform a window query on I_{DB} such that for each dimension in the 6-D space, the lower and upper boundary of the query window is $v - 2\triangle$ and $v + 2\triangle$, respectively, where v is the value of R_Q for this dimension, and obtain a result X which is a set of relative-distance representations in I_{DB} inside the query window. Note that each representation R_P in X is associated with an ID i and the point IDs of the 4 owners of R_P. Then, for each R_P in X, we insert a 2-tuple (R_Q, R_P) into $\mathscr{C}$.
- *Step c:* For each (R_Q, R_P) in $\mathscr{C}$, we perform the following step.
  - Firstly, let i be the ID of R_P.
  - Secondly, if i could be found in $\mathscr{R}$, we do nothing (since there is no need to process R_P again because the ID of its point cloud P is in the result $\mathscr{R}$). Otherwise (i.e., if i could not be found in $\mathscr{R}$), we do the following.
    - We perform a coarse transformation Θ based on (1) the 4 owners of R_Q and (2) the 4 owners of R_P.
    - Based on Θ, we perform a complete transformation Θ_o based on (1) all points of Q and (2) all points of point cloud P with ID i.
    - If $dist_{diff}(Q, P|\Theta_o) \leq \delta$, we insert the ID i into $\mathscr{R}$.

The following theorem shows the correctness of C_2O.

Theorem 1 (Correctness) *We are given a query point cloud Q and a non-negative real number δ. Let $\mathscr{R}$ be the set of database point clouds returned by our original query phase. Let $\mathscr{R}^*$ be the expected solution of our 3D object retrieval problem. Then, $\mathscr{R} = \mathscr{R}^*$.*

Proof To show $\mathscr{R} = \mathscr{R}^*$, (that is, our algorithm returns the correct result set), we just need to show that there is neither false positive (FP) nor false negative (FN) in our result. Firstly, it is obvious to see that our results do not contain any false positive, because in Step 2c, we only insert the database point clouds within the δ distance threshold of query Q into the result set. Next, to show there is no FN, we show that for any database point cloud $P \in \mathscr{R}^*$, $P \in \mathscr{R}$.

[1] R*-tree is a tree-like data structure to index points in a multi-dimensional space using minimum bounding boxes (MBB). It is very efficient to perform window queries or nearest neighbor queries using the R*-tree index by pruning many branches (represented by MBBs) in the R*-tree that cannot contain any query results.

By Lemma 5, since $dist(Q,P) \leq \delta$, the candidate set $\mathscr{C}_{q(1)}$ must contain a "desired" relative-distance representation $R_Q = rd(q_{(1)}|q_{(2)},q_{(3)},q_{(4)})$ that corresponds with an indexed relative-distance representation $R_P = rd(p_{(1)}|p_{(2)},p_{(3)},p_{(4)})$ in P. When we perform the window query for R_Q with query range $2\triangle$, R_P must also be one of the results by the definition of relative-distance representation (i.e., Equation 4). This is because, $\forall i \neq j \in [1,4]$, it holds that $\|q_{(i)},q_{(j)}\| - 2\triangle \leq \|p_{(i)},p_{(j)}\| \leq \|q_{(i)},q_{(j)}\| + 2\triangle$ (by Lemma 1). As a result, the 2-tuple (R_Q,R_P) (where R_P is associated with the ID of P) exists in $\mathscr{C}$. Next, in Step 2c, since we have included (R_Q,R_P) in $\mathscr{C}$, we will finally perform a complete transformation Θ_o between Q and P. Since we use Go-ICP [58] which ensures that Θ_o is the optimal transformation such that $dist_{diff}(Q,P|\Theta_o)$ is minimized, we will obtain the result that $dist_{diff}(Q,P|\Theta_o) = dist(Q,P) \leq \delta$. This indicates that the ID of P will be inserted into the returned set $\mathscr{R}$. □

5 Related Work

We first describe the relevancy to existing problems in Section 5.1 and then describe the relevancy to existing algorithms in Section 5.2.

5.1 Relevancy to Existing Problems

As described in Section 1, our 3D object retrieval problem is closely related to *registration* [7,18,50,37,25,26] and *similarity search* [14,36,55,59]. It is also related to *pattern matching* [48,13,20] and *object detection* [16,62,45] though there are some differences. Pattern matching can be regarded as one special case of our problem when there is *only one* database point cloud and one query point cloud and our distance function is replaced by another distance function. Object detection is to find a set of objects in 3D point clouds with models like deep-learning models which require some "labelled" datasets containing some objects "manually" marked as objects. However, object detection does not involve any query point cloud in its problem which is fundamentally different from us and could only detect objects with *known* shapes from the datasets. Our problem does not need "labelled" datasets and is flexible to any query object with an arbitrary shape, and thus our problem is different from object detection and is more challenging.

5.2 Relevancy to Existing Algorithms

Although no existing algorithms solve our 3D object retrieval problem exactly, there are some existing algorithms which can be adapted to our problem. They are *Super4PCS* [39] and *Go-ICP* [58].

Super4PCS adopts the well-known *4-point co-planar matching* scheme [7] to find the best transformation between two point clouds P and Q. It involves three steps, namely (1) the point-pair retrieval step (which is to find two random point pairs from Q, forming a 4-point structure Ψ, and to find a set S_1 of point pairs from P with pairwise distance equal to one of the random point pairs and another set S_2 of point pairs from P with pairwise distance equal to the other random point pair), (2) the 4-point structure search step (which, for each point pair in S_1 and each point pair in S_2, is to construct a 4-point structure based on these 2 point pairs and to insert this structure to a variable F if this structure is "similar" to Ψ) and (3) the transformation verification step (which, for each structure Φ in F, is to perform a coarse transformation on Ψ (for matching the space of Φ) and then to perform a locally optimized transformation on the *entire* point cloud Q (containing Ψ) (for matching the space of the database point cloud containing Φ) by a method like ICP [49]). Note that *Super4PCS* returns locally-optimized transformations only (not necessarily globally optimal transformation), since it generally follows a fundamental paradigm called RANSAC [28] that could find the best transformation between two point clouds in high chance (but not exactly) by repeatedly executing the above three steps a number of times, each with different random point pairs for testing.

Go-ICP can be regarded as the state-of-the-art algorithm to find the *globally optimal* transformation between two point clouds. In *Go-ICP*, each transformation is represented by 6 parameters and thus, the search space considered is the space containing all possible values from these parameters. It searches for the best transformation in this search space by a Branch-and-bound (BnB) search strategy where each iteration of the BnB search splits a search space being considered into 8 equal-sized subspaces until the optimal transformation is found. However, *Go-ICP* is costly, with $O(8^l)$ time complexity in the worst case where l is a data-dependent parameter denoting the maximum number of iterations in the BnB search (and is about 30 on average in our experiment). Note that although the original Go-ICP has high time cost, it will become much more efficient if an initial transformation Θ close to the optimal is first given. This is because, with this initial transformation, *Go-ICP* could find the exact optimal transformation by a fast sub-procedure in *Go-ICP*.

We adapt *Super4PCS* [39] and *Go-ICP* [58] as follows.

- (1) **Super4PCS-Adapt(NoIndex)**: We modify *Super4PCS* to form our exact approach as follows. Firstly, in the 4-point structure search step, we replace their heuristic error parameter by $\triangle$ (the upper bound of the correspondence distance for all query points) to determine whether two 4-point structures are "similar". Secondly, in the transformation verification step, after obtaining the coarse transformation results, we run the complete transformation and object retrieval steps of our C_2O algorithm for global optimality. Thirdly, since *Super4PCS* handles two point clouds only, we run our adapted algorithm between the query and each database point cloud.

– (2) **Super4PCS-Adapt(Index)**: Based on **Super4PCS-Adapt(NoIndex)**, we enhance the point-pair retrieval step by introducing a 1-dimensional index (e.g., B+-tree) to index all pairwise distances among all points in each database point cloud. Note that in all steps of **Super4PCS-Adapt(NoIndex)**, the only step we could improve with the index is the point-pair retrieval step and the improvement is based on the pairwise distance search (which leads us to propose to introduce a one-dimensional index). With this index, the two sets (i.e., S_1 and S_2) could be found more efficiently.
– (3) **GoICP-Adapt**: Since *Go-ICP* gives the *optimal* transformation between two point clouds only, we run *Go-ICP* for the query and each database point cloud. For each result, if the optimal distance is within δ, we include the corresponding database point cloud in the result set.

The non-index approach (i.e., **Super4PCS-Adapt(NoIndex)**) does not perform well (compared with our C_2O algorithm) because the 4-point search time cost is linear to the database size [39], but C_2O just takes $O(\log n + k_2)$ where k_2 is the size of the output. The index approach (i.e., **Super4PCS-Adapt(Index)**) does not perform well either (compared with C_2O) because the acceleration by the 1D index only affects the point-pair retrieval step, and the remaining steps for finding the matched 4-points are still time-consuming. Since the output size from the index could be as large as the database size, this algorithm is not efficient enough. Note that in our C_2O query phase, we have a similar step of finding all the relative-distance representations in the database. Our step only takes $O(\log n + k_2)$ time, where k_2 is very small and is equal to 40 on average in our experiment. The rationale of our improvement is that we represent and match a 4-point as a *whole* instead of handling it as two *separate* point-pairs in existing approaches. Finally, **GoICP-Adapt** does not perform well due to its high computation cost.

There are some existing algorithms solving the similarity search problem. They are PointNetVLAD [55], PCAN [59], feature-based descriptors [14,52,12,24] and 3D shape descriptors [36,33]. Two representative algorithms are PointNetVLAD [55] and PCAN [59] which are deep learning models trained on a number of 3D point clouds and are used to find the query point cloud efficiently with the trained deep learning models via embedding vectors as described in Section 1. However, both models do not consider any rotation/translation of the given query point cloud and thus, the results from these models are not accurate. To improve their accuracy, we adapt the *training* phase of each of these two models. To improve their accuracy, we adapt the *training* phase of each of these two models as follows. Given a training query point cloud, they need to find and label all the similar database point clouds to form training samples. To obtain this, we apply our distance measurement instead of their original similarity measurement. Moreover, methods of feature-based descriptors [14,52,12,24] form a *local descriptor* (which could be represented in the form of histogram) measuring some geometric features of neighboring points (e.g., the pairwise distances among these points) for each of the sampled key points from a point cloud, and finally match two point clouds with similar local descriptors. Unfortunately, they still give inaccurate results since the descriptors are easily affected by noise and two key point sets (which are *subsets* of the entire point clouds) with the same local descriptors cannot be guaranteed for two similar (entire) point clouds. Methods of 3D shape descriptors [36,33] either summarize a global descriptor from the above local descriptors of key points for a similarity search [36] (thus they still suffer from the same issues as feature-based descriptors), or form a rough representation of the entire point cloud using some heuristic functions (e.g., one based on spherical harmonics) and perform similarity search based on this representation [33] (thus they still cannot capture the 3D shape exactly).

6 Experiment

6.1 Setup

We conducted experiments on a machine with 2.66GHz CPU and 48G memory. The programming language is C++ without any CPU- or GPU-level parallelization.

6.1.1 Datasets

We used the well-structured 3D object dataset repository called *redwood* [5] (because this dataset is commonly used in the literature of 3D graphics [44,21,22]). We used two datasets from this repository, namely *Object* [22] and *Indoor* [44]. We used another dataset, namely *OS-MN40* [27], which is from the recent 3D object retrieval challenge [6]. Dataset *Object* involves 441 3D objects each of which has a small point cloud, dataset *Indoor* involves 5 objects each of which has a large point cloud representing a scene in an indoor environment, and dataset *OS-MN40* involves 9,487 3D CAD objects each of which has a small point cloud. We process each dataset as follows.

– In dataset *Object*, following the setting of [42,57,46], for each object, we formed a point cloud of size around 100 using quadric edge collapse decimation [54] (a seminal method implemented in MeshLab [23]). The diameter of the minimum bounding sphere covering each point cloud (simply called the *diameter* of this point cloud) is 3,000–6,000mm. We randomly chose 400 objects in dataset *Object* to form a new database dataset called *Object-DB*, serving for the database purpose, and chose the remaining 41 objects to form a new dataset *Object-OutsideDB* which will be used in the query generation (to be described later).

– In dataset *Indoor*, we chose three objects which are bedroom, boardroom and loft (with 2.5M, 4.5M and 3M points, respectively) to form a new database dataset called *Indoor-DB* and chose the remaining 2 objects to form a new dataset *Indoor-OutsideDB*. The total database size of *Indoor-DB* is 10M. The diameter of each point cloud is approximately 250,000mm.
– In dataset *OS-MN40*, 8,527 objects are used for the "collection" set, and the remaining 960 objects are used for the "query" set. Similar to dataset *Object*, we also formed a point cloud of size around 100 for each object in dataset *OS-MN40* using quadric edge collapse decimation. The diameter each point cloud is approximately 1,700mm. Based on this dataset, we also formed a database dataset called *OS-MN40-DB* which includes all the 8,527 objects in the "collection" set. The size of dataset *OS-MN40-DB* is 850K. Note that the objects in the "query" set are only used for the query purpose in this dataset (as described later).

Based on the database datasets, we would like to construct additional datasets for scalability test. In dataset *Object-DB*, since it contains 400 objects only, we create 5 datasets, namely *Object #1*, *Object #2*, *Object #3*, *Object #4* and *Object #5*, with the number of objects as 100, 1K, 10K, 100K and 1M, respectively. For the first dataset, we use 100 (out of 400) objects from dataset *Object-DB*. For the remaining 4 datasets, we generate them as follows. Since each of these datasets contains more than 400 objects, we generate additional objects with the following steps. We pick one object (or point cloud) P from the original dataset (i.e., *Object*) and create a new object by perturbing the coordinates of each point in P with distortion values generated according to Gaussian distribution with mean equal to 0 and standard deviation equal to 0.05 times the diameter of point cloud P. The sizes of the 5 generated datasets (i.e., n) in *Object-DB* are 12.2K, 118K, 1.18M, 11.8M and 117M, respectively. In dataset *Indoor-DB*, we create 3 datasets, namely *Indoor #1*, *Indoor #2* and *Indoor #3* with database size as 10K, 100K and 1M. Since each of these 3 datasets is smaller than dataset *Indoor-DB*, we sample all point clouds in dataset *Indoor-DB* using quadric edge collapse decimation [54] such that the database size of each of these 3 datasets is equal to the desired size. We then re-scale the three datasets to their diameters around 10,000mm, 30,000mm and 80,000mm, respectively, such that the average point-pairwise distance of each dataset is similar to that of dataset *Indoor-DB*. We also call dataset *Indoor-DB* as *Indoor #4*. For dataset *OS-MN40-DB*, we mainly study the effectiveness for retrieval accuracy with this dataset, and thus we do not construct additional datasets. The statistics of these datasets (together with dataset *OS-MN40-DB*) are summarized in Table 2.

Dataset	$\lvert\mathscr{P}\rvert$	Size (i.e., n)	Dataset	$\lvert\mathscr{P}\rvert$	Size (i.e., n)
Object #1	100	12.2K	***Indoor* #1**	3	10K
***Object* #2**	1K	118K	*Indoor* #2	3	100K
Object #3	10K	1.18M	*Indoor* #3	3	1M
Object #4	100K	11.8M	*Indoor* #4	3	10M
Object #5	1M	117M	*OS-MN40-DB*	8,527	850K

Table 2 Statistics about Datasets

6.1.2 Random Query Generation

We generate random queries as follows. For dataset *Object*, we randomly pick one point cloud as query from *Object-DB*. For dataset *Indoor*, we select one scene randomly from the 3 scenes in *Indoor-DB* and extract a random part from the selected scene as the query point cloud with diameter roughly equal to the target diameter as 1,000mm. We also vary this extraction diameter to 1,500mm, 2,000mm, 2,500mm and 3,000mm for our scalability tests. For dataset *OS-MN40*, we randomly pick one point cloud from its "query" set. Note that there is no identical object between its "query" set and its "collection" set (which is used to form its DB dataset), and thus the queries bear less similarity with the database objects, which makes the retrieval task more challenging. As such, dataset *OS-MN40* is particularly used to verify the effectiveness of the retrieval accuracy. For each obtained query point cloud Q in all datasets, we perform a random translation and rotation on Q, and to simulate varied levels of noise, we also introduce noise levels of 10%, 20%, 30% and 40% by performing a perturbation on each coordinate value of Q following Gaussian distribution with mean equal to 0 and standard deviation equal to 0.0025 times the diameter of Q [61]. In addition, we test different types of queries for dataset *Object* as follows. *(a) Non-existing and mixing types:* we randomly pick one point cloud from the rest 41 objects in dataset *Object* but outside *Object-DB* for query (called the "non-existing" queries) and we also mix the queries in *Object-DB* and the *non-existing* queries together with varied proportions (details can be found in [8]). *(b) Partial-matching types:* we extract a random part of a query point cloud Q in dataset *Object* such that the proportion of points in each extracted part over all points in Q (called *overlap*) is 25%, 50% and 75%, respectively. Note that the overlap of the original queries in dataset *Object* is 100%. For dataset *Indoor*, we form similar non-existing and mixing types of queries by extracting *non-existing* queries from the remaining 2 scenes, and form similar partial-matching types of queries by varying the overlap from 2% to 10%.

6.1.3 Algorithms

We include all the adapted existing algorithms (described in Section 5) as baselines for comparison: **Super4PCS-Adapt(NoIndex)** [39], **Super4PCS-Adapt(Index)** [39] and **GoICP-Adapt** [58]. Our proposed algorithm is denoted by $\mathbf{C_2O}$. In addition, we also include the two state-of-the-art deep learning algorithms for comparison, denoted as **PointNetVLAD** [55] and **PCAN** [59].

For each database/query point cloud, we build an R*-tree for some spatial queries like nearest neighbor queries and range queries (used in our proposed algorithm and our compared algorithms). Following the pre-processing steps of the deep learning algorithms [55,59], we split each database point cloud of dataset *Indoor* into a number of small point clouds using a voxel grid [29] of grid size equal to our default query diameter (to be introduced later) where each small point cloud (containing all the points within a cell) represents a portion of a database point cloud. For some experiments where the query diameter is varied for dataset *Indoor*, we set the grid size to different values accordingly to form different pre-processed *Indoor* datasets. Moreover, we follow [55,59] to mark a retrieved portion as correct if its center has its Euclidean distance to the center of the expected portion within the diameter of the query point cloud (that is, it is very close to the expected portion). Furthermore, since the deep learning algorithms are designed to retrieve the top-k similar objects (or portions) in the database, we return the top-k where k is set to the number of all the expected database point clouds (or portions) for a given query and δ. Due to the architecture nature of the deep learning algorithms requiring 128 points as input, we re-sample each database/query object (or portion) with exactly 128 points using quadric edge collapse decimation.

6.1.4 Factors

We vary the following factors: (1) r, (2) query point cloud size, (3) database size, (4) δ, (5) query noise percentage and (6) overlap. (1) By default, we set r to 1,200mm, 350mm and 150mm for datasets *Object*, *Indoor* and *OS-MN40*, respectively, which lead to the best (i.e., smallest) query time. (2) The default query diameter for dataset *Indoor* is 1,000mm. Unlike dataset *Indoor*, there is no need to vary query diameter for datasets *Object* or *OS-MN40* since a whole database object is specified as a query object. (3) The default dataset sizes are 118K, 10K, and 850K for datsets generated from *Object*, *Indoor* and *OS-MN40*. (4) Following [58], we set the default value of δ to be 3.2% (of the query diameter) for datasets *Object* and *Indoor*. For dataset *OS-MN40*, we set δ to be 20% (of the query diameter) since this setting gives accurate results of retrieving similar objects with the same type (as described later). (5) The default query noise percentage (as described in Section 6.1.2) is 10%. (6) The default overlap is 100% (2%) for dataset *Object* (*Indoor*).

6.1.5 Measurements

We have the following measurements: (1) the index building time, (2) the index size, (3) the query time, (4) the number of query relative-distance representation candidates, (5) the number of complete transformations and (6) the precision, recall and F-measure (showing how accurate the algorithms are). All measurements are straightforward. Note that the F-measure is defined to be the harmonic mean of the precision and recall, where the precision is defined to be the proportion of the expected results retrieved by an algorithm (i.e., the true positives) over all the retrieved results, and the recall is defined to be the proportion of the retrieved expected results over all the expected results. Each measurement is reported as an average of at least 100 random query executions.

In our experiments, we obtain the expected results of each query Q based on the distance defined in Equation 3. That is, the expected results are the database point clouds with distance to Q at most δ. Later, in Section 6.2.3, we show that by setting δ to the default values, our retrieval results are accurate and effective for the real-world retrieval problem.

6.2 Results

We show the results in the following parts. In Section 6.2.1, we study the design of our C_2O algorithm. In Section 6.2.2, we compare the efficiency of C_2O with other algorithms. In Section 6.2.3, we show the effectiveness of C_2O in terms of retrieval accuracy.

6.2.1 Study of Design of Our C_2O Algorithm

We conducted experiments about the design of $\mathbf{C_2O}$, i.e., choosing the best value of r (based on the smallest query time), choosing the regular tetrahedron (based on the high pruning power), choosing the strategy of selecting $q_{(1)}$ from Q (based on the smallest query time), studying the effectiveness of pruning relative-distance representations with a multi-dimensional index I_{DB}, and choosing R*-tree to implement I_{DB} (based on the smallest query time). Details could be found in our technical report [8].

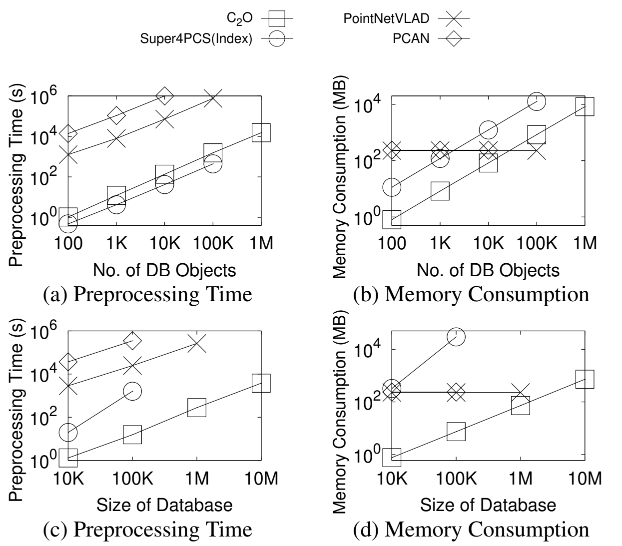


Fig. 8 Preprocessing Time and Memory Consumption for (a)&(b) Dataset *Object* and (c)&(d) Dataset *Indoor*

6.2.2 Comparison Among All Algorithms for Efficiency

In the following, we give the experimental results for the efficiency.

Preprocessing Efficiency: We study the preprocessing time and the memory consumption of index-based algorithms (i.e., $\mathbf{C_2O}$ and **Super4PCS-Adapt(Index)**) and deep learning algorithms (i.e., **PointNetVLAD** and **PCAN**). The preprocessing time and the memory consumption of an index-based algorithm refer to its index construction time and its index size, respectively. The preprocessing time and the memory consumption of deep learning algorithms refer to its training time and its deep learning model size. Figures 8(a) and (b) show the results about the preprocessing time and the memory consumption on dataset *Object*, respectively. In Figure 8(a), the preprocessing times of all algorithms increase with the number of database objects, while both deep learning algorithms need 3 orders of magnitude more time for preprocessing than $\mathbf{C_2O}$ due to costly training. Though $\mathbf{C_2O}$ has slightly longer preprocessing time, it consumes much smaller memory than **Super4PCS-Adapt(Index)** that has a too bulky index size to be executed on the 1M-object database. Figure 8(b) shows that the memory consumption of index-based algorithms increases with the number of database objects, while that of deep learning algorithms remains unchanged (since their model sizes are independent of the number of objects). As shown in Figures 8(c) and (d) and Table 3, $\mathbf{C_2O}$ achieves similarly superior preprocessing efficiency compared to baselines.

Effect of Size of Database: The query time of all non-deep learning algorithms increases with the number of database objects as shown in Figure 9(a). Our $\mathbf{C_2O}$ has the shortest query time (e.g., 4.6s when there are 1M database objects). The fastest baseline algorithm **Super4PCS-Adapt(Index)** has 2–3 orders of magnitude longer query time than $\mathbf{C_2O}$, although it slightly improves **Super4PCS-Adapt(NoIndex)** due to the simple 1D indexing structure to retrieve the point-pairs. **GoICP-Adapt** takes the longest query time due to its costly optimal transformation for all database objects. Note that the query times of the two deep learning algorithms (i.e., **PointNetVLAD** and **PCAN**) remain nearly unchanged at around 0.6s when the number of database objects varies because the time of searching the embedding vectors of all database objects in the K-D tree is very small and the increase of search time is insignificant as the number of database objects increases due to the efficient search of the K-D tree. Nevertheless, in our default setting with 1K objects, our $\mathbf{C_2O}$ is still much faster (i.e., in 0.032s) than deep learning algorithms. Later, we will show that the deep learning algorithms are inaccurate. It is worth mentioning that the result of **PointNetVLAD** (**PCAN**) are shown on the dataset with at most 10K (100K) objects, because for larger datasets, their preprocessing times are too long (e.g., more than 10^6s).

Effect of Size of Query: Figure 9(b) shows that the query times of all algorithms except deep learning algorithms increase with the size of the query (i.e., $|Q|$). When $|Q|$ increases, the parameter $\triangle$ $(= \delta|Q|^{1/2})$ also increases, resulting in increased query times for our algorithm and **Super4PCS**-related algorithms. Though $\mathbf{C_2O}$ shows a slightly larger increasing trend (since $\mathbf{C_2O}$ could generate a larger candidate set for a query point cloud with more points), $\mathbf{C_2O}$ still outperforms **Super4PCS-Adapt(Index)** by more than 1 order of magnitude. The query times of deep learning algorithms remain unchanged when the size of the query varies. This is because their model architectures require 128 points as input and different query sizes will be re-sampled to 128 points as an input, resulting in the nearly unchanged query times.

Effect of δ: Figure 9(c) shows that the query times of all algorithms except deep learning algorithms increase with δ. Our $\mathbf{C_2O}$ still outperforms all the non-deep learning algorithms.

Effect of Noise Percentage: As shown in Figure 9(d), the noise percentage does not affect the query times of all algorithms, and $\mathbf{C_2O}$ is still the fastest.

Query Results for Other Datasets: Our $\mathbf{C_2O}$ obtains similar superior efficiency as shown above for dataset *Indoor* and dataset *OS-MN40*), as shown in Figure 25 and Table 3, respectively. In particular, on the 1M-point-database of dataset *Indoor*, the query time of our $\mathbf{C_2O}$ is within 4.1s but the exact baselines need at least 657s (as shown in Figure 25(a)). For dataset *OS-MN40* (of size 850K with a larger δ $(= 20\%)$), we perform queries within 2.3s, while

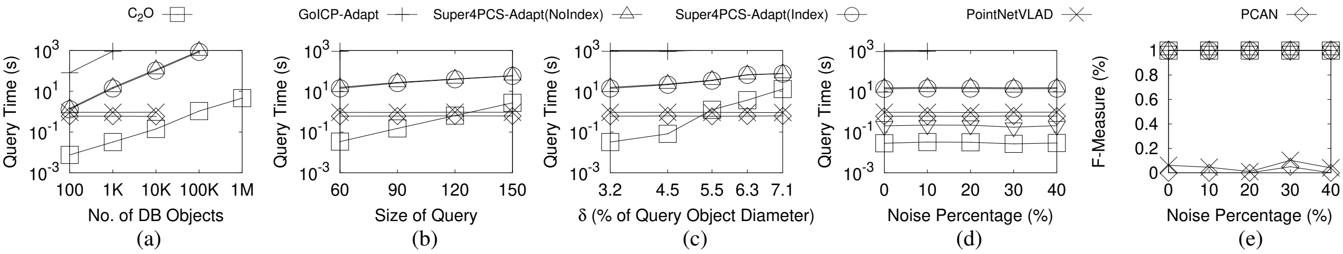


Fig. 9 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object*

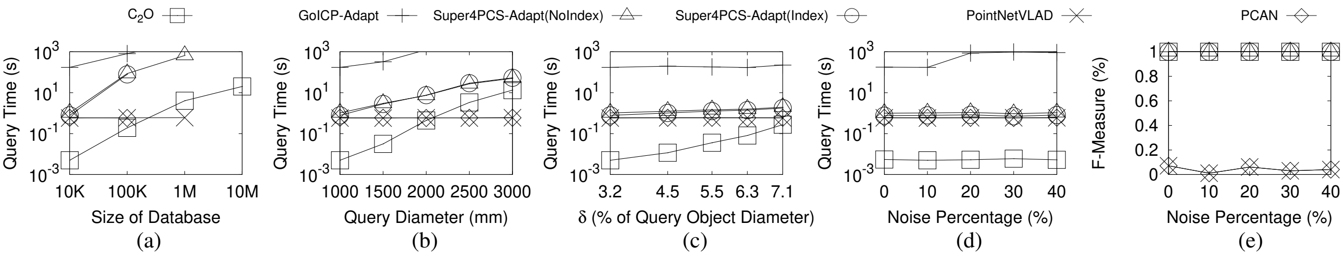


Fig. 10 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor*

Algorithm	Preprocessing Time	Memory Consumption	Query Time	F-Measure (Precision/Recall)
GoICP-Adapt	-	-	> 1000s	100% (100%/100%)
Super4PCS-Adapt(NoIndex)	-	-	307s	100% (100%/100%)
Super4PCS-Adapt(Index)	56.3s	825MB	282s	100% (100%/100%)
PointNetVLAD	1674s	131MB	1.12s	14.8% (13.6%/16.2%)
PCAN	8946s	127MB	1.39s	13.8% (11.5%/17.1%)
C_2O	78.2s	45MB	2.27s	100% (100%/100%)

Table 3 Comparison of Algorithms on Dataset *OS-MN40-DB*

the exact baselines need more than 282s (as shown in Table 3). It is worth mentioning that the database dataset of *OS-MN40* (i.e., dataset *OS-MN40-DB*) and the 100-object database dataset of *Object* (i.e., dataset *Object #1*) consist of the point clouds from the original dataset only (without any perturbation of points). For both databases, our $\mathbf{C_2O}$ is 100x faster than the best exact baseline.

Results for Other Query Types: We conducted experiments with other types of queries as described in Section 6.1.2 (i.e., the second-, third-, fourth- and fifth-type queries). We obtained similar experimental results. It is worth mentioning that the query time of $\mathbf{C_2O}$ for the queries involving more objects outside the database (i.e., the second- and fourth-type queries) are slightly larger than those on other queries because more database objects have to be processed to find similar objects. But, the query times are all within 11.6s for $\mathbf{C_2O}$ (even for 1M database objects). See [8] for details.

6.2.3 *Effectiveness of* $\mathbf{C_2O}$ *for Retrieval Accuracy*

In the following, we show the effectiveness of $\mathbf{C_2O}$ in terms of retrieval accuracy. First, we observe that our $\mathbf{C_2O}$ (and any other exact algorithm) always returns query results with 100% F-measure (i.e., the query results are exactly the same

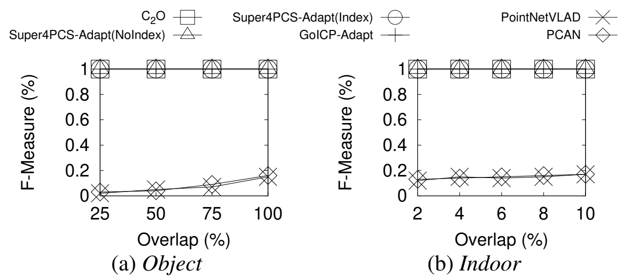


Fig. 11 Effect of Overlap on F-Measure for Dataset *Object* and *Indoor*

as the ground-truth). This verifies the ability of our $\mathbf{C_2O}$ to answer the 3D object retrieval problem exactly. However, the F-measures of deep learning algorithms are only around or less than 15% (with both precision and recall $< 25\%$), because the deep learning algorithms do not consider any rotation/translation, and thus they return irrelevant objects (i.e., incorrect results) in most cases. Moreover, the 100% F-measure result of $\mathbf{C_2O}$ is consistent in more challenging cases (e.g., high noise percentage, as shown in Figures 9(e) and 25(e), and low overlap, as shown in Figure 11), while the deep learning algorithms perform even worse (e.g., around 5% F-measure when overlap $= 25\%$ for dataset *Object*, as

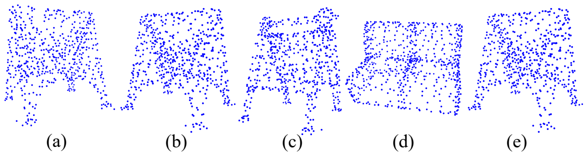


Fig. 12 Case Study of Dataset *Object* with Query Object (a) as Standing Sign ($\delta = 3.2\%$)

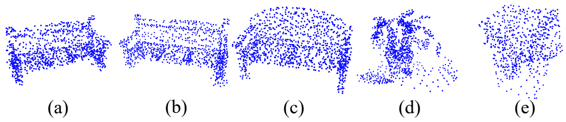


Fig. 13 Case Study of Dataset *OS-MN40* with Query Object (a) as Bench ($\delta = 20\%$)

shown in Figure 11(a)). Next, we further show our effectiveness in retrieval accuracy by case studies.

Case Studies: We conducted a number of case studies about the results returned by our $\mathbf{C_2O}$ and the deep learning algorithms. Figure 12(a) shows the query object Q (i.e., a standing sign) for dataset *Object* (note that Q could be in arbitrary position and orientation, which is simulated by a random translation and rotation in our experiments). When setting δ to our default value of this dataset (i.e., 3.2% of the query diameter), our $\mathbf{C_2O}$ returns the exact object (as shown in Figure 12(b)) as well as another standing sign that is very similar to Q (as shown in Figure 12(c)). Note that both objects have distance to Q (as defined in Equation 3 in the form of the average L_2-norm) within the default δ. However, the deep learning algorithm **PointNetVLAD** returns two objects where one of them is correct but the other is an irrelevant object (as shown in Figures 12(d) and (e)). Another case study for dataset *OS-MN40* is shown in Figure 13. In this dataset, there is no exact matching of the query object, and thus we set the default δ to 20% of the query diameter so that the similar database objects with the same type could be retrieved. In this case study, the similar benches to the query (as shown in Figure 13(a)) are returned (with two examples shown in Figures 13(b) and (c)), while most results returned by **PointNetVLAD** are irrelevant objects with other types (with two examples shown in Figures 13(d) and (e)).

6.2.4 Summary of Results

We verify the superior efficiency of our proposed $\mathbf{C_2O}$ algorithm for object retrieval queries, which generally outperforms the existing accurate algorithms by 1-3 orders of magnitude for various experimental configurations. In dataset *Object* with 1M objects (over 100M points), our algorithm obtains efficient and superior performance (e.g., within 5 seconds), while none of the existing accurate algorithms handle it in reasonable time (e.g., less than 1000 seconds). Moreover, our algorithm returns accurate results with 100% F-measure value, but the existing deep learning algorithms obtain at most around 15% F-measure value.

7 Conclusion

In this paper, we studied the problem about efficient 3D object retrieval which have many applications. We propose our C_2O algorithm, which, in most of our experiments, performs up to 1–2 orders of magnitude faster than all adapted algorithms in the literature. Moreover, C_2O guarantees 100% accurate results but some adapted algorithms do not. Some possible future work could be the top-k version and the dynamic version of 3D object retrieval.

References

1. (2022). URL https://www.yeggi.com/
2. (2023). URL https://blog.google/products/maps/street-view-15-new-features/
3. (2023). URL https://developers.google.com/ar/reference/java/com/google/ar/core/PointCloud
4. (2023). URL https://www.azury.one/en/technology/
5. (2023). URL http://redwood-data.org/
6. (2023). URL https://www.shrec.net/
7. Aiger, D., Mitra, N.J., Cohen-Or, D.: 4-points congruent sets for robust pairwise surface registration. In: ACM transactions on graphics (TOG), vol. 27, p. 85. Acm (2008)
8. Authors, A.: On efficient 3d object retrieval (technical report) (2023). URL https://github.com/anonym62371/F458812606D3F73A
9. Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B.: The r*-tree: an efficient and robust access method for points and rectangles. In: Acm Sigmod Record, vol. 19, pp. 322–331. Acm (1990)
10. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM **18**(9), 509–517 (1975)
11. Besl, P.J., McKay, N.D.: Method for registration of 3-d shapes. In: Sensor fusion IV: control paradigms and data structures, vol. 1611, pp. 586–606. International Society for Optics and Photonics (1992)
12. van Blokland, B.I., Theoharis, T.: An indexing scheme and descriptor for 3d object retrieval based on local shape querying. Computers & Graphics **92**, 55–66 (2020)
13. Brass, P.: Exact point pattern matching and the number of congruent triangles in a three-dimensional pointset. In: European Symposium on Algorithms, pp. 112–119. Springer (2000)
14. Bustos, B., Keim, D.A., Saupe, D., Schreck, T., Vranić, D.V.: Feature-based similarity search in 3d object databases. ACM Computing Surveys (CSUR) **37**(4), 345–387 (2005)
15. Chen, C.S., Hung, Y.P., Cheng, J.B.: Ransac-based darces: A new approach to fast automatic registration of partially overlapping range images. IEEE Transactions on Pattern Analysis and Machine Intelligence **21**(11), 1229–1234 (1999)
16. Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3d object detection network for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1907–1915 (2017)
17. Chen, X., Vizzo, I., Läbe, T., Behley, J., Stachniss, C.: Range image-based lidar localization for autonomous vehicles. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 5802–5808. IEEE (2021)

18. Chen, Y., Medioni, G.: Object modelling by registration of multiple range images. Image and vision computing **10**(3), 145–155 (1992)
19. Cheung, K.M., Jun, W., Lightsey, G., Lee, C.: Single-satellite real-time relative positioning for moon and mars (2019)
20. Chew, L.P., Goodrich, M.T., Huttenlocher, D.P., Kedem, K., Kleinberg, J.M., Kravets, D.: Geometric pattern matching under euclidean motion. Computational Geometry **7**(1-2), 113–124 (1997)
21. Choi, S., Zhou, Q.Y., Koltun, V.: Robust reconstruction of indoor scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5556–5565 (2015)
22. Choi, S., Zhou, Q.Y., Miller, S., Koltun, V.: A large dataset of object scans. arXiv preprint arXiv:1602.02481 (2016)
23. Cignoni, P., Callieri, M., Corsini, M., Dellepiane, M., Ganovelli, F., Ranzuglia, G.: Meshlab: an open-source mesh processing tool. In: Eurographics Italian chapter conference, vol. 2008, pp. 129–136 (2008)
24. Dimou, D., Moustakas, K.: Fast 3d scene segmentation and partial object retrieval using local geometric surface features. Transactions on Computational Science XXXVI: Special Issue on Cyberworlds and Cybersecurity pp. 79–98 (2020)
25. Dong, J., Cai, Z., Du, S.: Improvement of affine iterative closest point algorithm for partial registration. IET Computer Vision **11**(2), 135–144 (2016)
26. Elbaz, G., Avraham, T., Fischer, A.: 3d point cloud registration for localization using a deep neural network auto-encoder. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4631–4640 (2017)
27. Feng, Y., Gao, Y., Zhao, X., Guo, Y., Bagewadi, N., Bui, N.T., Dao, H., Gangisetty, S., Guan, R., Han, X., et al.: Shrec'22 track: Open-set 3d object retrieval. Computers & Graphics **107**, 231–240 (2022)
28. Fischler, M.A., Bolles, R.C.: Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM **24**(6), 381–395 (1981)
29. Foley, J.D., van Dam, A., Hughes, J.F., Feiner, S.K.: Spatial-partitioning representations; surface detail. Computer Graphics: Principles and Practice (1990)
30. Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3354–3361. IEEE (2012)
31. Golovinskiy, A., Kim, V.G., Funkhouser, T.: Shape-based recognition of 3d point clouds in urban environments. In: 2009 IEEE 12th International Conference on Computer Vision, pp. 2154–2161. IEEE (2009)
32. Goparaju, A., Iyer, K., Bone, A., Hu, N., Henninger, H.B., Anderson, A.E., Durrleman, S., Jacxsens, M., Morris, A., Csecs, I., et al.: Benchmarking off-the-shelf statistical shape modeling tools in clinical applications. Medical Image Analysis **76**, 102,271 (2022)
33. Kazhdan, M., Funkhouser, T., Rusinkiewicz, S.: Rotation invariant spherical harmonic representation of 3 d shape descriptors. In: Symposium on geometry processing, vol. 6, pp. 156–164 (2003)
34. Keselman, L., Woodfill, J.I., Grunnet-Jepsen, A., Bhowmik, A.: Intel (r) realsense (tm) stereoscopic depth cameras. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 1267–1276. IEEE (2017)
35. Kim, S., Kim, S., Lee, D.E.: Sustainable application of hybrid point cloud and bim method for tracking construction progress. Sustainability **12**(10), 4106 (2020)
36. Körtgen, M., Park, G.J., Novotni, M., Klein, R.: 3d shape matching with 3d shape contexts. In: The 7th central European seminar on computer graphics, vol. 3, pp. 5–17. Citeseer (2003)
37. Li, H., Hartley, R.: The 3d-3d registration problem revisited. In: 2007 IEEE 11th international conference on computer vision, pp. 1–8. IEEE (2007)
38. Meagher, D.: Geometric modeling using octree encoding. Computer graphics and image processing **19**(2), 129–147 (1982)
39. Mellado, N., Aiger, D., Mitra, N.J.: Super 4pcs fast global pointcloud registration via smart indexing. In: Computer Graphics Forum, vol. 33, pp. 205–215. Wiley Online Library (2014)
40. Mohamad, M., Ahmed, M.T., Rappaport, D., Greenspan, M.: Super generalized 4pcs for 3d registration. In: 2015 International Conference on 3D Vision, pp. 598–606. IEEE (2015)
41. Mohamad, M., Rappaport, D., Greenspan, M.: Generalized 4-points congruent sets for 3d registration. In: 2014 2nd international conference on 3D vision, vol. 1, pp. 83–90. IEEE (2014)
42. Murali, A., Mousavian, A., Eppner, C., Paxton, C., Fox, D.: 6-dof grasping for target-driven object manipulation in clutter. In: 2020 IEEE International Conference on Robotics and Automation (ICRA), pp. 6232–6238. IEEE (2020)
43. Pagani, A., Gava, C.C., Cui, Y., Krolla, B., Hengen, J.M., Stricker, D.: Dense 3d point cloud generation from multiple high-resolution spherical images. In: VAST, pp. 17–24 (2011)
44. Park, J., Zhou, Q.Y., Koltun, V.: Colored point cloud registration revisited. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 143–152 (2017)
45. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum pointnets for 3d object detection from rgb-d data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 918–927 (2018)
46. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: Pointnet++: Deep hierarchical feature learning on point sets in a metric space. arXiv preprint arXiv:1706.02413 (2017)
47. Qinghan, B., Sihua, D., Chenguang, L., Ze, Q.: Application of bim in the creation of prefabricated structures local parameterized component database. Architecture and Engineering **4**(2), 13–21 (2019)
48. de Rezende, P.J., Lee, D.: Point set pattern matching ind-dimensions. Algorithmica **13**(4), 387–404 (1995)
49. Rusinkiewicz, S., Levoy, M.: Efficient variants of the icp algorithm. In: 3dim, vol. 1, pp. 145–152 (2001)
50. Rusu, R.B., Blodow, N., Beetz, M.: Fast point feature histograms (fpfh) for 3d registration. In: 2009 IEEE International Conference on Robotics and Automation, pp. 3212–3217. IEEE (2009)
51. Rusu, R.B., Blodow, N., Marton, Z., Soos, A., Beetz, M.: Towards 3d object maps for autonomous household robots. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3191–3198. IEEE (2007)
52. Rusu, R.B., Blodow, N., Marton, Z.C., Beetz, M.: Aligning point cloud views using persistent feature histograms. In: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3384–3391. IEEE (2008)
53. Samirbhai, M.D., Chen, S.: A star pattern recognition technique based on the binary pattern formed from the fft coefficients. In: 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1–5. IEEE (2018)
54. Tarini, M., Pietroni, N., Cignoni, P., Panozzo, D., Puppo, E.: Practical quad mesh simplification. In: Computer Graphics Forum, vol. 29, pp. 407–418. Wiley Online Library (2010)
55. Uy, M.A., Lee, G.H.: Pointnetvlad: Deep point cloud based retrieval for large-scale place recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2018)
56. Van Krevelen, D., Poelman, R.: Augmented reality: Technologies, applications, and limitations. Vrije Univ. Amsterdam, Dep. Comput. Sci (2007)
57. Xu, D., Anguelov, D., Jain, A.: Pointfusion: Deep sensor fusion for 3d bounding box estimation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 244–253 (2018)
58. Yang, J., Li, H., Jia, Y.: Go-icp: Solving 3d registration efficiently and globally optimally. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1457–1464 (2013)

59. Zhang, W., Xiao, C.: Pcan: 3d attention map learning using contextual information for point cloud based retrieval. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 12,436–12,445 (2019)
60. Zhang, Z.: Microsoft kinect sensor and its effect. IEEE multimedia **19**(2), 4–10 (2012)
61. Zhou, Q.Y., Park, J., Koltun, V.: Fast global registration. In: European Conference on Computer Vision, pp. 766–782. Springer (2016)
62. Zhou, Y., Tuzel, O.: Voxelnet: End-to-end learning for point cloud based 3d object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4490–4499 (2018)

A Extended Details of Algorithm C_2O

A.1 Extended Details of Donut Representation

In Section 4.1, we introduced the steps of forming tetrahedron, which generate a tetrahedron close to a regular tetrahedron (represented by $p_{(1)}$, $p_{(2)}$, a and b). Now, we give a lemma to formally show that the tetrahedron represented by point $p_{(1)}$, $p_{(2)}$, a and b is a regular tetrahedron.

Lemma 6 *The tetrahedron represented by $p_{(1)}$, $p_{(2)}$, a and b is regular.*

Proof Since m is the mid-point between $p_{(1)}$ and $p_{(2)}$, $ma \perp mp_{(1)}$ and $\|ma\| = r' = \frac{\sqrt{3}}{2}\|p_{(1)}, p_{(2)}\|$, we could derive that $\triangle p_{(1)}ap_{(2)}$ is a regular triangle. Since point b is decided by rotating ma around point m inside the perpendicular plane of $mp_{(1)}$, it holds that $mb \perp mp_{(1)}$ and $\|m,a\| = \|m,b\|$. Similarly, we know that $\triangle p_{(1)}bp_{(2)}$ is also a regular triangle. It left to show that $\|a,b\| = \|p_{(1)}, p_{(2)}\|$ such that all the edges of tetrahedron formed by point $p_{(1)}$, $p_{(2)}$, a and b are equal. Since $\theta = \arccos 1/3$, by the law of cosines, it is easy to derive that $\|a,b\| = \sqrt{2r'^2 - 2r'^2\cos\theta} = \frac{2\sqrt{3}}{3}r' = \|p_1, p_2\|$.

A.2 Extended Details of Why Bound $\triangle$ Is Tight

In the following lemma, we show that the bound $\triangle = \delta|Q|^{1/2}$ introduced in Section 4.2 is tight by showing that, in Lemma 1, it is possible that $\|q', p\| = \triangle$.

Lemma 7 *Consider a database point cloud P. There exists a query point cloud Q such that Q satisfies the following 2 conditions. (1) $dist(Q,P) \leq \delta$ and (2) there exists a point q' in Q' such that $\|q', p\| = \triangle$, where Q' is the point cloud optimally transformed from Q (wrt P) and p is the correspondence point of q' on P.*

Proof We construct a query point cloud Q based on a database point cloud P in $\mathscr{P}$ as follows. Let Q be $\varnothing$ initially. For each point p in P except an arbitrarily selected point, we create a point q with the same coordination as p, i.e., $q = p$, and then insert q into Q. After that, we insert into Q a point q' such that the distance between q' and the nearest point of q' in P is $\triangle$. It can be verified that $dist(Q,P) = \sqrt{\triangle^2/|Q|} = \delta$ (i.e., condition (1) is satisfied). Also, considering the point q' whose correspondence point p on p (i.e., the nearest point of q' in P), then $\|q', p\|$ is exactly $\triangle$ (i.e., condition (2) is satisfied).

A.3 Extended Details of Rotation and Translation Invariant Property of Candidate Set Generation

In this section, we give the following lemma to show that the output candidate set generated by the steps of constructing the candidate set of relative-distance representations introduced in Section 4.2.4 has the translation-invariant and rotation-invariant properties.

Lemma 8 *Consider a query point cloud Q. Let $q_{(1)}$ be a point in Q. Let $\Theta = (R,t)$ be a transformation consisting of a rotation matrix R and a translation vector t. Consider another query point cloud Q' such that $Q' = T_\Theta(Q)$. Let $q'_{(1)}$ be a point in Q'. Let $\mathscr{C}_{q_{(1)}}$ ($\mathscr{C}'_{q'_{(1)}}$) be the output of the steps to find the set of candidate relative-distance representations in Q (Q') with the starting point $q_{(1)}$ ($q'_{(1)}$). If $q'_{(1)} = T_\Theta(q_{(1)})$, then $\mathscr{C}_{q_{(1)}} = \mathscr{C}'_{q'_{(1)}}$.*

Proof We show that $\mathscr{C}_{q_{(1)}} = \mathscr{C}'_{q'_{(1)}}$ by the following two claims: (1) for each $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) \in \mathscr{C}_{q_{(1)}}$, there exists a relative-distance representation $rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)}) \in \mathscr{C}'_{q'_{(1)}}$ such that $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$ and (2) for each $rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)}) \in \mathscr{C}'_{q'_{(1)}}$, there exists a relative-distance representation $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) \in \mathscr{C}_{q_{(1)}}$ such that $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$.

For the first claim, consider that $q'_{(2)} = T_\Theta(q_{(2)})$, $q'_{(3)} = T_\Theta(q_{(3)})$ and $q'_{(4)} = T_\Theta(q_{(4)})$. Next, we need to show that $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$ and $rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)}) \in \mathscr{C}'_{q'_{(1)}}$.

Firstly, since for each $i \in [1,4]$, $q'_{(i)}$ is the transformed point of $q_{(i)}$ by the same transformation Θ, it is obvious that the distance between each pair-wise distance among the four points remains unchanged. Therefore, by the definition of relative-distance representation (i.e., Equation 4), it is easy to get $rd(q_{(1)}|q_{(2)}, q_{(3)}, q_{(4)}) = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$.

Secondly, we perform the steps to find the set of candidate relative-distance representations with the same Q, r and $\triangle$ starting from $q_{(1)}$ and $q'_{(1)}$, respectively, as two *instances* denoted by $\mathscr{I}$ and $\mathscr{I}'$, respectively. We continue our proof by showing that we obtain the sets of same points in $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$ for $\mathscr{I}$ and $\mathscr{I}'$, and in the following we say that $q \in Q$ and $q' \in Q'$ are the same points if $q' = T_\Theta(q)$.

(1) Let q' (q'') be the *ball-NN* operation result we obtain (i.e., $ball\text{-}NN(q_{(1)}, r + 2\triangle|Q)$) for $\mathscr{I}$ ($\mathscr{I}'$). Since Q' is obtained by transforming every point in Q with the same transformation Θ, we know that the distance between every other point to $q_{(1)}$ remains unchanged. Therefore, the *ball-NN* operation result also remains unchanged (i.e., $q'' = T_\Theta(q')$). And then, we can obtain the same set of points inside $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$ for $\mathscr{I}$ and $\mathscr{I}'$. Further, let $S'_{(2)}$ be the set we obtain in Step (i) for $\mathscr{I}'$. By the above result, $S'_{(2)}$ must contain $q'_{(2)}$ which equals to $T_\Theta(q_{(2)})$.

(2) In Step (i)(1), we focus on the case of $q_{(2)}$ ($q'_{(2)}$) for $\mathscr{I}$ ($\mathscr{I}'$). Let m', $\mathbf{n}'$, r'' be the result of the mid-point between $q'_{(1)}$ and $q'_{(2)}$, the vector from $q'_{(2)}$ to $q'_{(1)}$ and $\frac{\sqrt{3}}{2}\|q'_{(1)}, q'_{(2)}\|$, respectively, for $\mathscr{I}'$. Then, it is also obvious that $m' = T_\Theta(m)$, $\mathbf{n}' = T_\Theta(\mathbf{n})$ and $r'' = r'$. As such, we also obtain a donut D' for $\mathscr{I}'$ which contains exactly the same points as D. Again, due to rigid transformation, the distance between every point in Q to D remains unchanged. Thus, we get the same resultant point for the *donut-NN* operation and thus we can obtain the same set of points inside $dbz(D, [0, dist(q', D) + 4\triangle])$ for $\mathscr{I}$ and $\mathscr{I}'$. Let $S'_{(3)}$ be the set we obtain in Step (i)(1) for $\mathscr{I}'$. By the above result, $S'_{(3)}$ must contain $q'_{(3)}$ which equals to $T_\Theta(q_{(3)})$.

(3) In Step (i)(1)(I), we focus on the case of $q_{(3)}$ ($q'_{(3)}$) for $\mathscr{I}$ ($\mathscr{I}'$). Let a' and b' be the two virtual points for $\mathscr{I}'$. Since again all the points

remain their distance to D, then we have $a' = T_\Theta(a)$ and $b' = T_\Theta(b)$. Thus, we can obtain the same set of points inside $sbz(b, [0\ \|b, q''\| + 4\triangle])$ for $\mathscr{I}$ and $\mathscr{I}'$. Let $S'_{(4)}$ be the set we obtain in Step (i)(1)(I) for $\mathscr{I}'$. By the above result, $S'_{(4)}$ must contain $q'_{(4)}$ which equals to $T_\Theta(q_{(4)})$. Finally, we will obtain a relative-distance representation $R = rd(q'_{(1)}|q'_{(2)}, q'_{(3)}, q'_{(4)})$ and include it in the result set $\mathscr{C}'_{q_{(1)}}$.

Till now, we have proved the first claim. Consider the reverse transformation of Θ (i.e., Θ^{-1}). Then, we have $Q = T_{\Theta^{-1}}(Q')$ and $q_{(1)} = T_{\Theta^{-1}}(q'_{(1)})$. Thus, we can directly prove the second claim by swapping Q with Q', swapping $q_{(1)}$ with $q'_{(1)}$ and substituting Θ with Θ^{-1}.

A.4 Extended Query Complexity Analysis

In this section, we analyze the time complexity of our relative-distance representation candidate generation step in the query phase (introduced Section 4.3.2) and the complexity of the number of generated candidates wrt the size of query.

Recall the steps of constructing the candidate set in Section 4.2.4 starting from a point $q_{(1)}$. Given a query point cloud Q and the starting point $q_{(1)}$, we first find set $S_{(2)}$ containing points in Q inside the sphere bounding zone $sbz(q_{(1)}, [r - 2\triangle, \|q_{(1)}, q'\| + 4\triangle])$, which takes $O(\log|Q| + |S_{(2)}|)$ time. Then, for each point $q_{(2)}$ in $S_{(2)}$, we find set $S_{(3)}$ containing points in Q inside the donut bounding zone $dbz(D, [0,\ dist(q', D) + 4\triangle])$, which takes $O(\log|Q| + |S_{(3)}|)$ time. Next, for each point $q_{(3)}$ in $S_{(3)}$, we find set $S_{(4)}$ containing points in Q inside the sphere bounding zone $sbz(b, [0\ \|b, q''\| + 4\triangle])$, which takes $O(\log|Q| + |S_{(4)}|)$ time. We denote M to be the average number of points in the generated sets (i.e., $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$) among all the steps. Thus, the time complexity of constructing the candidate set from $q_{(1)}$ is approximately $O(M^3 \log|Q|)$, and the total number of generated candidate representations in Q is approximately $O(M^3)$.

It is clear that each of the generated sets (i.e., $S_{(2)}$, $S_{(3)}$ and $S_{(4)}$) contains query points inside a "thin" zone whose depth is $O(\triangle)$. But, in the worst case, M could be as large as $O(|Q|)$. Correspondingly, the total number of generated candidate representations could be as large as $O(|Q|^3)$ in the worst case. However, in practice, the bound $\triangle$ $(= \delta|Q|^{1/2})$ is small. Thus, M is small and the number of query candidates is also small. In a typical experimental setting (e.g., with the database size 1M and the query size 100), M is about 15 on average and the number of query candidates is about 400 on average. Figure 16(b) (which is introduced later in the additional experimental results) shows the number of query candidates with the varied size of query. It can be seen that, in practice, the growth of the number of query candidates is much smaller than the worst-case cubic complexity wrt the query size.

Next, since we find the query point with the smallest candidate set by an extensive search, the overall time complexity of candidate generation (i.e., Step 1 of the query phase in Section 4.3.2) is $O(|Q|M^3 \log|Q|)$ in the worst case. Then, for each candidate, we find a set of database relative-distance representations in the index I_{DB} by a window query. Let R be the expected number of representations in this set. With efficient R*-tree index to implement I_{DB}, the expected time to complete the window query is $O(\log n + R)$, where n is the database size. At this point, the time complexity for performing all the window queries is thus $O(M^3(\log n + R))$.

Finally, for each window query result (totally $O(M^3 R)$ results in expectation), we perform a complete transformation using Go-ICP [58]. Although the practical time for Go-ICP with a initial transformation close to optimal is fast, the worst time complexity could still be $O(8^l)$, where l is a data dependent parameter in Go-ICP. Therefore, the overall time complexity of our query phase is $O(M^3(|Q|\log|Q| + \log n + 8^l R))$.

A.5 Extended Details of a Heuristic Acceleration Strategy

Although the two steps of our C_2O query phase introduced in Section 4.3.2 are efficient (using index I_{DB}), we want to speed up our process with a strategy called "Relevancy Filtering" (RF).

The major idea of the RF strategy is given as follows. Note that after we choose *only one* point $q_{(1)}$ from Q in Step 1, we obtain the 2-tuple candidate set $\mathscr{C}$ of $q_{(1)}$ in Step 2b where $\mathscr{C}$ contains a number of 2-tuples each in the form of (R_Q, R_P). We could use the RF strategy to prune some of the 2-tuples in $\mathscr{C}$ with the following two steps to be introduced.

The first step in the RF strategy is performed just after Step 1 (and just before Step 2). Specifically, we generate a set $\mathscr{H}$ of h additional points in Q, namely $q'_{(1),1}, q'_{(1),2}, \ldots, q'_{(1),h}$, such that these h additional points are the h nearest points of $q_{(1)}$ in Q. For each point in $\mathscr{H}$, says $q'_{(1),j}$ where $j \in [1,h]$, we could also obtain a 2-tuple candidate set $\mathscr{C}'_j$ of $q'_{(1),j}$ (instead of $q_{(1)}$) which has the same procedure as Step 2b.

The second step in the RF strategy is performed just after Step 2b. Before we describe this second step, we first introduce a concept of "closeness" and a concept of "relevancy". Given two points $p_{(1)}$ and $p'_{(1)}$ in P and two points $q_{(1)}$ and $q'_{(1)}$ in Q, we say that pair $(p_{(1)}, p'_{(1)})$ is $\triangle$-*close* to pair $(q_{(1)}, q'_{(1)})$ if the following condition holds.

$$\|q_{(1)}, q'_{(1)}\| - 2\triangle \leq \|p_{(1)}, p'_{(1)}\| \leq \|q_{(1)}, q'_{(1)}\| + 2\triangle \tag{5}$$

In other words, when $\triangle$ is small, the distance between $p_{(1)}$ and $p'_{(1)}$ is "roughly" equal to the distance between $q_{(1)}$ and $q'_{(1)}$. If the distance between $q_{(1)}$ and $q'_{(1)}$ is small, the distance between $p_{(1)}$ and $p'_{(1)}$ is small. This $\triangle$-*closeness* relationship is the key idea for our second step which is to prune some 2-tuples in the candidate set $\mathscr{C}$. Consider a 2-tuple (R_Q, R_P) in $\mathscr{C}$. When $\triangle$ is small and the distance between $q_{(1)}$ and $q'_{(1)}$ is small, if pair $(p_{(1)}, p'_{(1)})$ is not $\triangle$-close to pair $(q_{(1)}, q'_{(1)})$, we know that the distance between $p_{(1)}$ and $p'_{(1)}$ is large. We could prune this 2-tuple since we expect that the distance between two query points (i.e., $q_{(1)}$ and $q'_{(1)}$) is "roughly" equal to the distance between two "matched" database points (i.e., $p_{(1)}$ and $p'_{(1)}$).

Given a 2-tuple (R_Q, R_P) in the 2-tuple candidate set $\mathscr{C}$ of a point $q_{(1)}$ in Q and a 2-tuple (R'_Q, R'_P) in the 2-tuple candidate set of another point $q'_{(1)}$ in Q, we say that R_P is *relevant* to R'_P wrt $(q_{(1)}, q'_{(1)})$ if (1) the ID of R_P is equal to the ID of R'_P and (2) for the first owner of R_P, namely $p_{(1)}$, and the first owner of R'_P, namely $p'_{(1)}$, pair $(p_{(1)}, p'_{(1)})$ is $\triangle$-close to pair $(q_{(1)}, q'_{(1)})$. In other words, we know that (1) the two relative-distance representations (i.e., R_P and R'_P) are in the same database point cloud and (2) the distance between the first owner of R_P and the first owner of R'_P is small if the distance between $q_{(1)}$ and $q'_{(1)}$ is small (and $\triangle$ is small).

In the above definition, we define the relevancy of a representation to another representation. Next, we overload the concept of relevancy to define the relevancy of a representation to a 2-tuple candidate set. Given (1) a 2-tuple (R_Q, R_P) in the 2-tuple candidate set $\mathscr{C}$ of a point $q_{(1)}$ in Q and (2) the 2-tuple candidate set $\mathscr{C}'$ of another point $q'_{(1)}$ in Q, we say that R_P is *relevant* to $\mathscr{C}'$ wrt $(q_{(1)}, q'_{(1)})$ if there exists a 2-tuple (R'_Q, R'_P) in $\mathscr{C}'$ such that R_P is *relevant* to R'_P wrt $(q_{(1)}, q'_{(1)})$.

With the concept of "closeness" and the concept of "relevancy", we are ready to describe this second step. This second step is to remove each 2-tuple in the form of (R_Q, R_P) from $\mathscr{C}$ if this 2-tuple does *not* satisfy the *complete relevancy* condition. Given (R_Q, R_P) in the candidate set $\mathscr{C}$ (of $q_{(1)}$), (R_Q, R_P) is said to satisfy the *complete relevancy* condition if for each candidate set $\mathscr{C}'_j$ (of point $q'_{(1),j}$) where $j \in [1,h]$, R_P is relevant to $\mathscr{C}'_j$ wrt $(q_{(1)}, q'_{(1),j})$. The reason is given as follows. Suppose that (R_Q, R_P) does not satisfy the complete relevancy condition. As long as we can find one point $q'_{(1),j}$ in $\mathscr{H}$ such that R_P is not relevant to the candidate set $\mathscr{C}'_j$ of $q'_{(1),j}$ wrt $(q_{(1)}, q'_{(1),j})$, for each 2-tuple

$(R'_{Q,j}, R'_{P,j})$ in $\mathscr{C}'_j$, R_P is not relevant to $R'_{P,j}$ wrt $(q_{(1)}, q'_{(1),j})$, and thus, we cannot obtain the "$\triangle$-closeness" relationship between $(p_{(1)}, p'_{(1),j})$ and $(q_{(1)}, q'_{(1),j})$ where $p_{(1)}$ $(p'_{(1),j})$ is the first owner of R_P $(R'_{P,j})$. It is worth mentioning that for the database point cloud P with the same ID as R_P, if $dist(Q,P) \leq \delta$, then for each $j \in [1,h]$, there exists a 2-tuple $(R'_{Q,j}, R'_{P,j})$ in $\mathscr{C}'_j$ such that (1) the ID of $R'_{P,j}$ is also equal to the ID of P and (2) we can obtain the "$\triangle$-closeness" relationship between $(p_{(1)}, p'_{(1),j})$ and $(q_{(1)}, q'_{(1),j})$ where $p_{(1)}$ $(p'_{(1),j})$ is the first owner of R_P $(R'_{P,j})$. Therefore, we know that "$dist(Q,P) \leq \delta$" does not hold for P, and thus, (R_Q, R_P) can be pruned.

In the following, we present a lemma to show the correctness of using the RF strategy. Specifically, let $\mathscr{C}_{RF}$ be the resulting candidate set of the RF strategy (i.e., $\mathscr{C}_{RF}$ contains all the 2-tuples in $\mathscr{C}$ that are not pruned). For a database point cloud P, if $dist(Q,P) \leq \delta$, $\mathscr{C}_{RF}$ will still contain a tuple, says (R_Q, R_P), such that each owner of R_P is the correspondence point of an owner of R_Q on P.

Lemma 9 *Consider a query point cloud Q and a database point cloud P. Let $q_{(1)}$ be a point in Q whose correspondence point on P is $p_{(1)}$. Let $p_{(2)}, p_{(3)}$ and $p_{(4)}$ be the second, third and fourth owners of the relative-distance representation of $p_{(1)}$, respectively. Let $\mathscr{C}_{RF}$ be the resulting candidate set after performing the steps of the RF strategy with the starting point $q_{(1)}$. If $dist(Q,P) \leq \delta$, then there exists a 2-tuple (R_Q, R_P) in $\mathscr{C}_{RF}$, such that $R_P = rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ and $p_{(i)}$ is the correspondence point of $q_{(i)}$ on P for $i \in [2,4]$, where $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$ are the second, third and fourth owners of R_Q, respectively.*

Proof Let $\mathscr{C}$ be the 2-tuple candidate set before we perform the second step of the RF strategy (i.e., before we prune some 2-tuples with the RF strategy). By the proof of Theorem 1, we know that there exists a 2-tuple (R_Q, R_P) in $\mathscr{C}$, such that $R_P = rd(p_{(1)}|p_{(2)}, p_{(3)}, p_{(4)})$ and $p_{(i)}$ is the correspondence point of $q_{(i)}$ on P for $i \in [2,4]$, where $q_{(2)}$, $q_{(3)}$ and $q_{(4)}$ are the second, third and fourth owners of R_Q, respectively. Now, it is only left to show that (R_Q, R_P) will not be pruned in the RF strategy. That is, (R_Q, R_P) satisfies the *complete relevancy* condition.

Consider an arbitrary point in $\mathscr{H}$, says $q'_{(1),j}$, which is used to obtain the 2-tuple candidate set $\mathscr{C}'_j$ (note that we do not perform any pruning on $\mathscr{C}'_j$). Since $q'_{(1),j}$ is an arbitrary point in $\mathscr{H}$, we then just need to show that R_P is relevant to $\mathscr{C}'_j$ wrt $(q_{(1)}, q'_{(1),j})$. Let $p'_{(1),j}$ be the correspondence point of $q'_{(1),j}$ on P. Let $p'_{(2),j}, p'_{(3),j}$ and $p'_{(4),j}$ be the second, third and fourth owners of the relative-distance representation of $p'_{(1),j}$, respectively. Since $dist(Q,P) \leq \delta$, identically, we know that there also exists a 2-tuple $(R'_{Q,j}, R'_{P,j})$ in $\mathscr{C}'_j$, such that $R'_{P,j} = rd(p'_{(1),j}|p'_{(2),j}, p'_{(3),j}, p'_{(4),j})$ and $p'_{(i),j}$ is the correspondence point of $q'_{(i),j}$ on P for $i \in [2,4]$, where $q'_{(2),j}$, $q'_{(3),j}$ and $q'_{(4),j}$ are the second, third and fourth owners of $R'_{Q,j}$, respectively. According to the definition of the relevancy of R_P to $\mathscr{C}'_j$ wrt $(q_{(1)}, q'_{(1),j})$, we can complete the proof by showing that R_P is relevant to $R'_{P,j}$ wrt $(q_{(1)}, q'_{(1),j})$.

The first condition is that R_P and $R'_{P,j}$ have the same ID, which is obvious since they both equal to the ID of P. The second condition is that pair $(p_{(1)}, p'_{(1)})$ is $\triangle$-close to pair $(q_{(1)}, q'_{(1)})$ (i.e., Equation 5 is satisfied). Since $p_{(1)}$ $(p'_{(1)})$ is the correspondence point of $q_{(1)}$ $(q'_{(1)})$ on P, Equation 5 is satisfied.

B Additional Experimental Results

We show our additional experimental results in the following parts. In Section B.1, we show the detailed results of studying the design of our C_2O algorithm. In Section B.2 and Section B.3, we show the additional results for datasets *Object* and *Indoor*, respectively.

B.1 Detailed Results of Studying the Design of Our C_2O Algorithm

Study of Parameter in our C_2O framework: We studied the major parameter in our C_2O framework, namely r (i.e, the side length of a tetrahedron). For the default setting of dataset *Object*, we build our $\mathbf{C_2O}$ index with varied radius r ranging from 500mm to 1,500mm. We take the average values to report experimental results.

In this experiment, after we set r to a value, we obtain an index based on a number of tetrahedra constructed in our $\mathbf{C_2O}$ index. Based on each index, we could issue queries described before and measure the query time. As illustrated in Figure 14(a), when r increases, the query time decreases first, reaches the minimum query time when r reaches around 1,200mm and increases after that. The reason is that smaller tetrahedra (with smaller r values) tend to trigger more (expensive) complete transformations in the database, which is consistent with existing observations [40] (since it is very likely that a small given tetrahedron is "similar" to a lot of small tetrahedra due to the *micro-view* (or too detailed view) from this given tetrahedron, resulting in a non-distinguishable tetrahedron). However, when r is very large, the cost of spatial operations (e.g. queries finding *ball-NN*) becomes higher and thus, the query time is larger. According to this, we set r = 1,200mm leading to the best-performing $\mathbf{C_2O}$ for the *Object* databases, and for dataset *Indoor*, we set r = 350mm following the similar trends in our experiments.

Study of 4-point Structure Compared with 3-point Structure: In this paper, we use the concept of regular tetrahedron to construct our donut representation. This concept involves a structure consisting of 4 points, which could lead to significantly better differentiating power than using 3-point structure [7]. We verified this conclusion by showing that the 4-point structure outperforms 3-point structure dramatically in both the number of complete transformations and query time.

We compared **Super4PCS-Adapt(Index)** (the best existing algorithm using 4-point structure [7]) and our $\mathbf{C_2O}$ algorithm with the adapted state-of-the-art 3-point structure approach [15] (denoted by **3PCS-Adapt(Index)**). Specifically, (1) we randomly select 3 points from Q to form a query 3-point structure Γ, (2) we find a set G of candidate 3-point structures in all database point clouds such that each candidate has similar structure as Γ within error parameter $\triangle$ (which is similar to the modification of **Super4PCS-Adapt(Index)**), (3) for each candidate 3-point structure Λ in G, we perform the coarse transformation based on Λ and Γ, and then perform complete transformation and object retrieval steps based on the coarse transformation result. Notably, we also apply the one-dimensional index built for **Super4PCS-Adapt(Index)** to accelerate the above Step (2), which also includes a same point-pair retrieval step in the middle.

As shown in Figure 14(b), the number of complete transformations of the existing 4-point algorithm (i.e., **Super4PCS-Adapt(Index)**) continue to outperform that of **3PCS-Adapt(Index)** by around 2 orders of magnitude as parameter δ increases. As a result, as shown in Figure 14(c), **Super4PCS-Adapt(Index)** also needs around 2 orders of magnitude less query time than **3PCS-Adapt(Index)**. It is worth mentioning that our $\mathbf{C_2O}$ algorithm even significantly outperforms **Super4PCS-Adapt(Index)** using the donut representation.

Study of Regularity of Tetrahedron: Next, we study why the regularity of a tetrahedron, the major principle used in our C_2O algorithm, is used. We also used the same experimental setup as the above experiment to build our $\mathbf{C_2O}$ index. In this experiment, the regularity of a tetrahedron could be described by the SD (Standard Deviation) ratio of side length which is defined to be the SD over the 6 side lengths divided by the average side length. A more regular tetrahedron tends to have similar side lengths, resulting in a smaller SD. Here, adopting the (relative) ratio (i.e., the SD divided by the average side length) instead of the absolute SD value is to study the regularity (which is relative in nature).

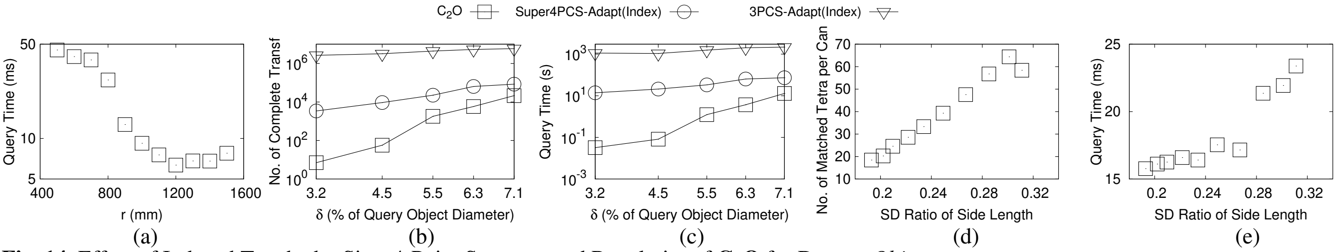


Fig. 14 Effect of Indexed Tetrahedra Size, 4-Point Structure and Regularity of $\mathbf{C_2O}$ for Dataset *Object*

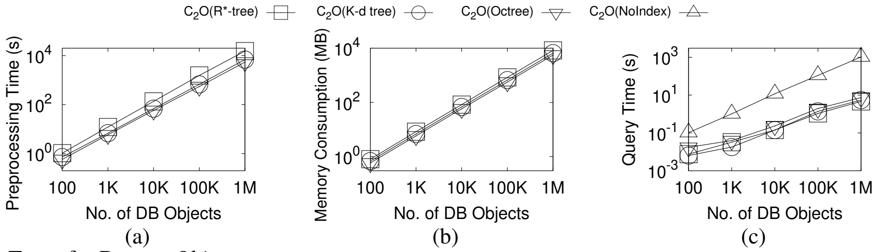


Fig. 15 Effect of Index Types for Dataset *Object*

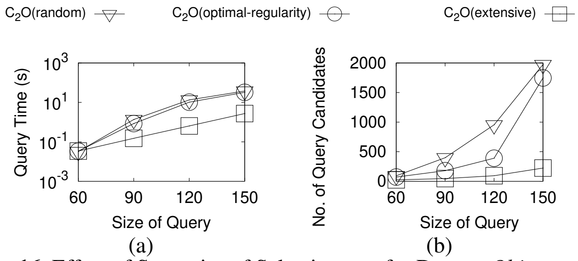


Fig. 16 Effect of Strategies of Selecting $q_{(1)}$ for Dataset *Object*

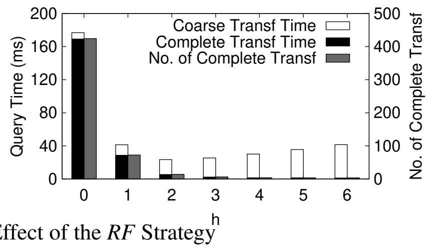


Fig. 17 Effect of the *RF* Strategy

We measure the number of database tetrahedra matched for each query tetrahedron candidate for each indexed tetrahedron.

Figure 14(d) shows that, in general, the number of database tetrahedra matched for each query tetrahedron candidate increases when the SD ratio increases. This means that less regularity (larger SD ratio) leads to more matched tetrahedra to be verified, leading to a larger query time as shown in Figure 14(e). This could justify our strategy of using more regular tetrahedra (where the SD ratio is smaller).

Study of Index Types: Since our C_2O algorithm can allow different types of multi-dimensional index to implement I_{DB}, we test three types of commonly used multi-dimensional index, namely R*-tree [9], k-d tree [10] and octree [38]. The three variants are indicated by $\mathbf{C_2O}$**(R*-tree)**, $\mathbf{C_2O}$**(K-d tree)** and $\mathbf{C_2O}$**(Octree)**, respectively. Note that for the octree variant, we use the first three dimensions in each 6-dimensional relative-distance representation for indexing, since the octree structure will have poor query performance when generalizing to higher dimensions. We also include a baseline here, indicated by $\mathbf{C_2O}$**(NoIndex)**, for our C_2O algorithm without building any multi-dimension index for efficient search (and thus the search is implemented by linear scanning all the relative-distance representations in the database).

As shown in Figure 15, the three variants have similar performance for all the related measurements (i.e., index building time, index size and query time), and they all outperform the baseline $\mathbf{C_2O}$**(NoIndex)** in query performance. Particularly, $\mathbf{C_2O}$**(Octree)** has the (slightly) best index building time and (slightly) smallest index size, while $\mathbf{C_2O}$**(R*-tree)** has the (slightly) slowest index building time and (slightly) largest index size. However, $\mathbf{C_2O}$**(R*-tree)** has the best query efficiency among the three variants, due to its superior performance of organizing points in the 6-dimension space. Compared with the baseline $\mathbf{C_2O}$**(NoIndex)**, using the R*-tree index leads to two orders of magnitude improvement on query time when the database scales to 1M objects. Due to its superior query efficiency, we use the R*-tree index as the default implementation of our C_2O algorithm.

Study of Strategies of Selecting $q_{(1)}$: In the query phase of C_2O, we use a strategy of selecting $q_{(1)}$ which is to extensively search for the point $q_{(1)}$ (among all points in Q) such that the number of the generated candidate set is the smallest. To verify the effectiveness of this strategy (which is denoted as $\mathbf{C_2O}$**(extensive)**), we compared it with another two strategies. The first strategy (which is denoted as $\mathbf{C_2O}$**(random)**) is to randomly pick a point in Q as $q_{(1)}$. The second strategy is based on the following heuristic. We form a *regularity measurement* of a point q in Q based on the *r-surrounding set* of q in Q where r is the radius parameter to construct our C_2O index. The r-surrounding set of q in Q is defined to be a set containing q itself and 3 other points in Q where these 3 points are the nearest to the surface of the sphere centered at q with radius r. Our regularity measurement of a point q in Q is defined to be the SD ratio of the tetrahedron formed by the r-surrounding set of q in Q. Intuitively, if the regularity measurement of q in Q is larger, then the r-surrounding set of q in Q can form a tetrahedron which is has a more regular shape. This could lead to the result of stronger pruning power. Therefore, the second strategy is to find the point in Q (and assign it to $q_{(1)}$) such that the regularity measurement is optimal (i.e., the smallest) among all points in Q by computing the regularity measurement of each point in Q. We denote the second stragy as $\mathbf{C_2O}$**(optimal-regularity)**.

As shown in Figure 16(a), when the size of query increases, all the three strategies need longer time to run the query. Except in the case of the smallest query size, the extensive search strategies achieves much better query efficiency than the other two strategies, which indicates better scalability. Recall that the number of candidates of query relative-distance representations is an important factor influencing the query time. As shown in Figure 16(b), the number of candidates for $\mathbf{C_2O}$**(extensive)** is much smaller than the other two strategies (since it always find the smallest candidate set), and thus $\mathbf{C_2O}$**(extensive)** is the most efficient.

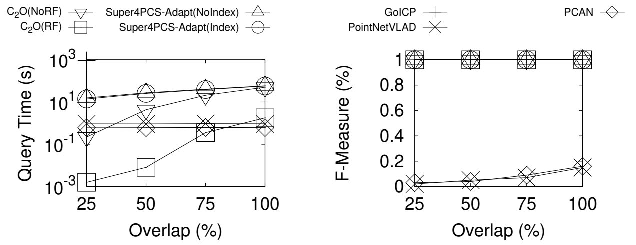


Fig. 18 Effect of Overlap for Dataset *Object*

Effect of Relevancy Filtering (RF) Strategy: In Section A.5, we propose the *RF* strategy to reduce the number of complete transformations by selecting h additional nearby query points for pruning. We now show how much query time this strategy improves and what value h to select by varying h from 0 (our algorithm without RF) to 6.

As illustrated in Figure 17, the *RF* strategy improves the overall query time, including both the coarse transformation time and the complete transformation time, significantly. Although the coarse transformation time (white bar) increases with h (since we perform more coarse transformation operations), the complete transformation time (black bar) decreases dramatically since the number of complete transformations (grey bar) decreases by 1–2 orders of magnitudes. However, when $h > 2$, the coarse transformation time begins to dominate the total query time, causing the total query time to increase slightly. We thus fix h to be 2 in the rest of the experiments for the best performance. For the remaining experimental results, and we use $\mathbf{C_2O(RF)}$ to denote our algorithm with the RF strategy. We also include the original algorithm (denoted by $\mathbf{C_2O(NoRF)}$) to fully show the effectiveness of the RF strategy.

B.2 Additional Results on Dataset *Object*

B.2.1 Effect of Overlap

We vary the overlap between query and database point clouds from 25% to 100%. As shown in Figure 18(b), $\mathbf{C_2O(RF)}$ is the most efficient among all exact algorithms and still performs accurately. In lower-overlap cases (e.g., overlap = 25%), the F-measure of deep learning algorithms is even smaller (e.g., around 5%), because it is more difficult for deep learning algorithms to capture the shape of query objects when the overlap is low. Nevertheless, $\mathbf{C_2O(RF)}$ always has 100% F-measure. The above findings are also discussed previously in Section 6.2.3. Moreover, Figure 18(a) shows that $\mathbf{C_2O(RF)}$ obtains the best efficiency among all exact algorithms, which is similar to our results of efficiency performance.

B.2.2 Results of Different Types of Queries

In this part, we demonstrate the comparison among all algorithms of dataset *Object* for different types of queries as described in Section 6.1.2. Specifically, the results for the second-type, third-type, fourth-type and fifth-type queries are reported in Figure 21, 22, 23 and 24, respectively.

The result we obtain for each of the other query types is similar to that the first type of queries (as shown in Figure 9). For the types of queries which contain more query objects outside the database (i.e., the second-type queries 100% of which are outside the database, and the fourth-type queries 80% of which are outside the database), we observe that all the non-deep learning algorithms use slightly more time to execute the queries (as shown in Figure 21 and 23, respectively) than the other types, but our proposed algorithms still have the

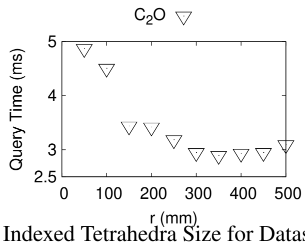
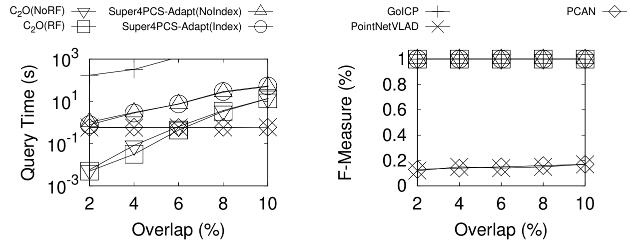


Fig. 19 Effect of Indexed Tetrahedra Size for Dataset *Indoor*



(a) (b)

Fig. 20 Effect of Overlap for Dataset *Indoor*

shortest query times among all the non-deep learning algorithms for all the query types. Specifically, in the default setting, when the type of queries is switched from the first-type to the second-type, the query time of our $\mathbf{C_2O(RF)}$ algorithm increases from 0.032s to 0.04s, while the query time of the fastest existing non-deep learning algorithm (i.e., **Super4PCS-Adapt(Index)**) increases from 13.9s to 22s.

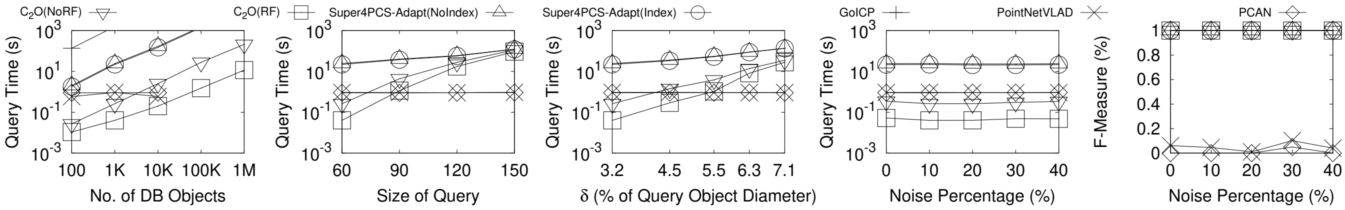
B.3 Additional Results on Dataset *Indoor*

In this part, we report the additional experimental results on dataset *Indoor*.

First, we show the query time of our algorithm $\mathbf{C_2O}$ with the effect of r in Figure 19. We set r ranging from 50mm to 500mm for dataset *Indoor*. We observe the similar trend as for dataset *Object* (as shown in Figure 14(a)). Specifically, the query time first drops when r increases from 50mm to 350mm and then increases slightly when r is larger than 350mm. Therefore, we set $r = 350$mm for all the experiments on dataset *Indoor* since it leads to the smallest query time.

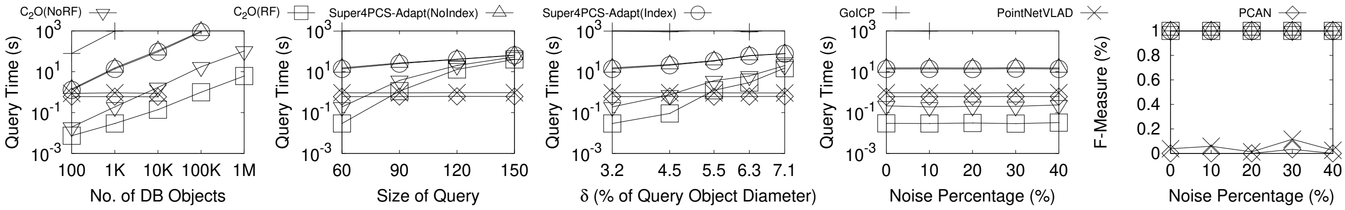
Then, we present results for varying the overlap in Figure 20, which also show similar results as in dataset *Object*. Note that the overlap between the default query (with diameter 1,000mm) and the database scene is only 2%, which verifies our capability of addressing a partial matching problem where the query is only a small part inside the large database scene. In this setting, since the query diameter is still in a reasonable range, we ensure that the matched results are still the meaningful part of the database scene (instead of noise).

We also run queries on dataset *Indoor* with all types of queries for all the factors we study. The results are reported in Figure 25, 26, 27, 28 and 29 for the first-type, second-type, third-type, fourth-type and fifth-type queries, respectively. All the results are similar to those on dataset *Object*.



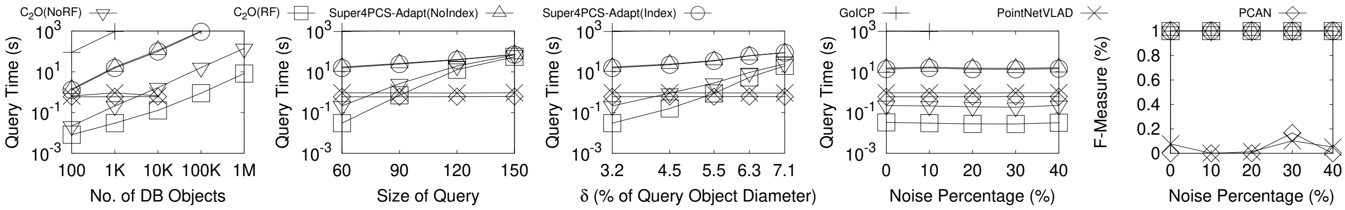
(a) (b) (c) (d) (e)

Fig. 21 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object* with the Second-type Queries



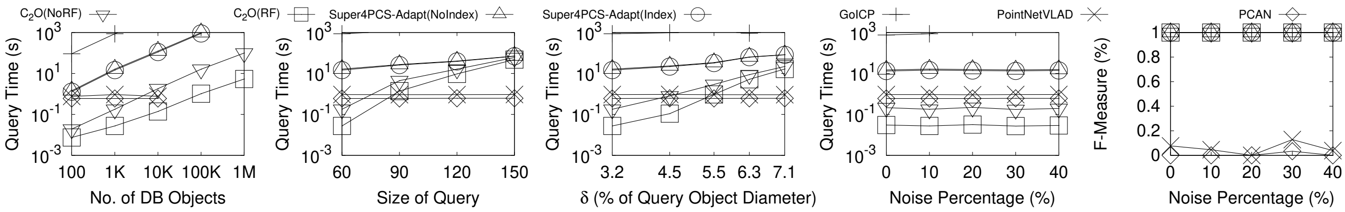
(a) (b) (c) (d) (e)

Fig. 22 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object* with the Third-type Queries



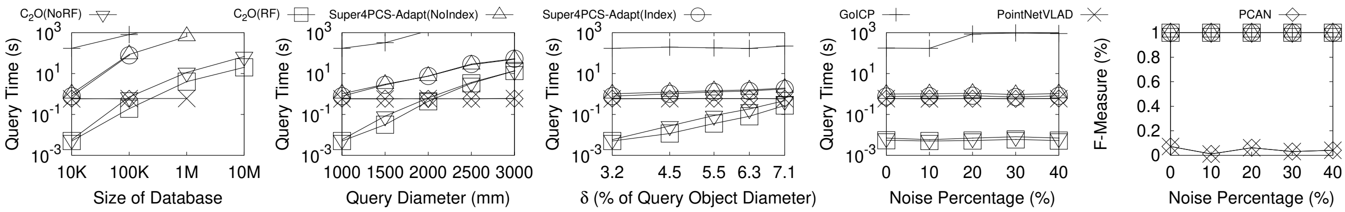
(a) (b) (c) (d) (e)

Fig. 23 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object* with the Fourth-type Queries



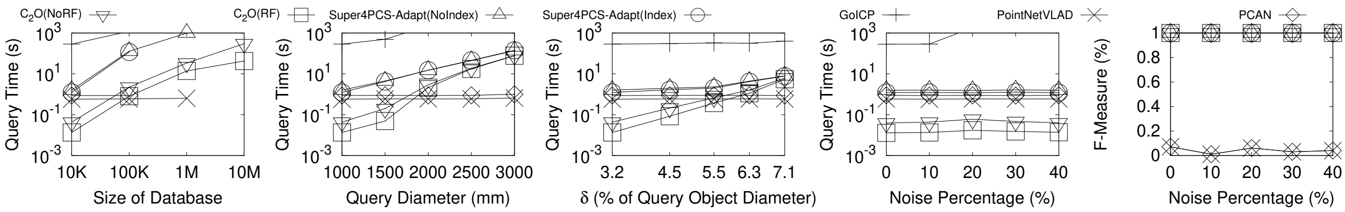
(a) (b) (c) (d) (e)

Fig. 24 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Object* with the Fifth-type Queries



(a) (b) (c) (d) (e)

Fig. 25 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the First-type Queries



(a) (b) (c) (d) (e)

Fig. 26 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the Second-type Queries

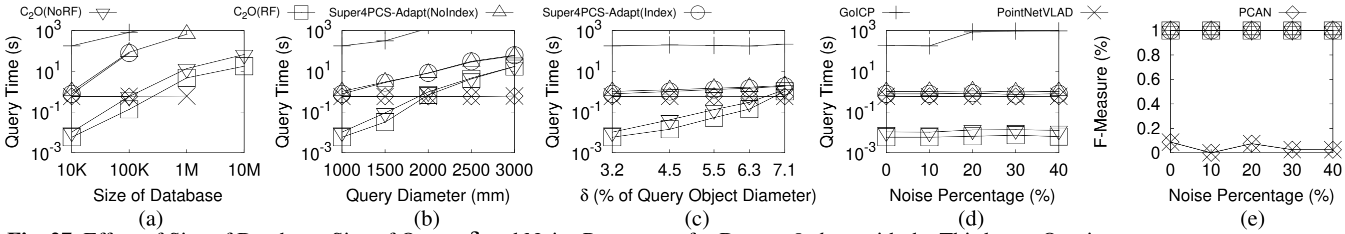


Fig. 27 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the Third-type Queries

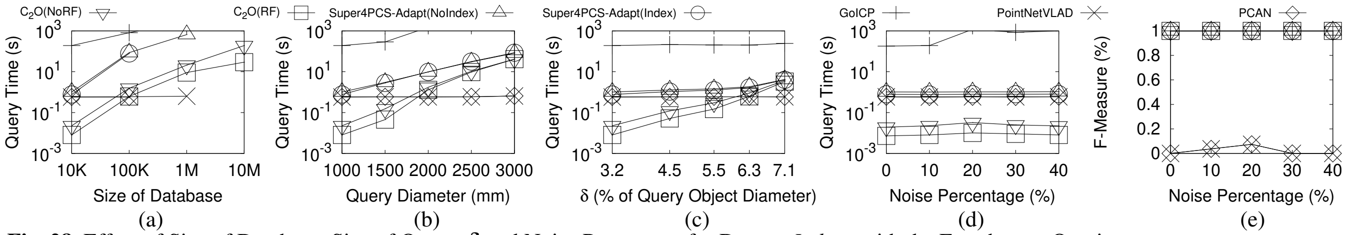


Fig. 28 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the Fourth-type Queries

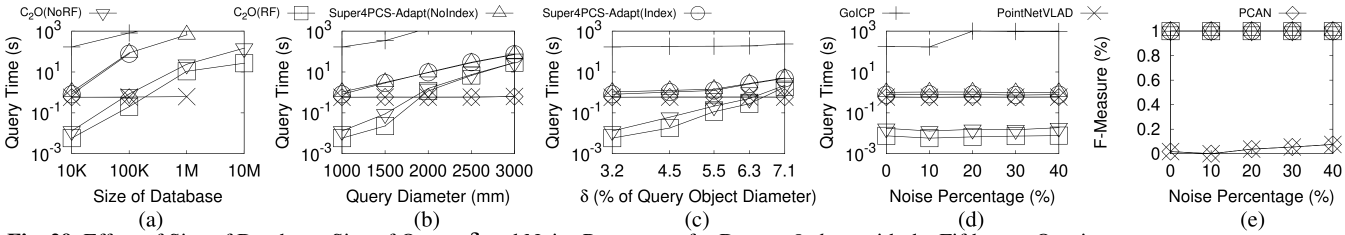


Fig. 29 Effect of Size of Database, Size of Query, δ and Noise Percentage for Dataset *Indoor* with the Fifth-type Queries